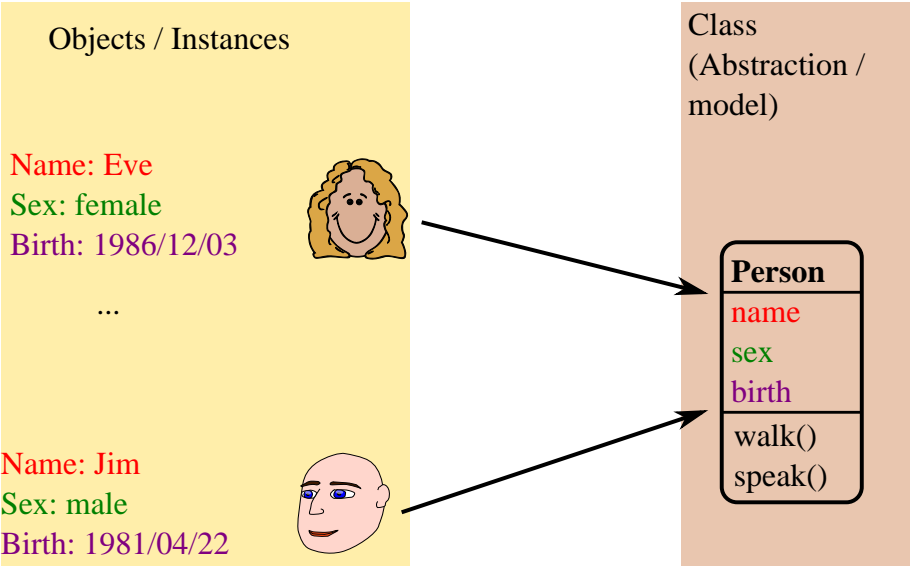


Instances of a Class



General class structure

Person	The classes name
name sex birth	Attributes
walk() speak()	Methods

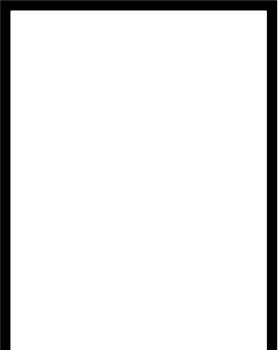
What's a class anyway?

In object oriented languages **classes**:

- are blueprints for objects.
- contain attributes and methods.
- allow for implementation hiding.
- allow for tailored access to methods and attributes.

Rectangle objects

width=20



height=28

width=28



dashed bord

A class describing rectangles

```
public class Rectangle {  
    int width;  
    int height;  
  
    // solid or dashed:  
    boolean hasSolidBorder;  
}
```



The diagram shows a class box for 'Rectangle'. The box is divided into three horizontal sections. The top section is yellow and contains the class name 'Rectangle' in bold black text. The middle section is light brown and contains the attributes 'width', 'height', and 'hasSolidBorder' in black text. The bottom section is light green and is currently empty.

Rectangle

width

height

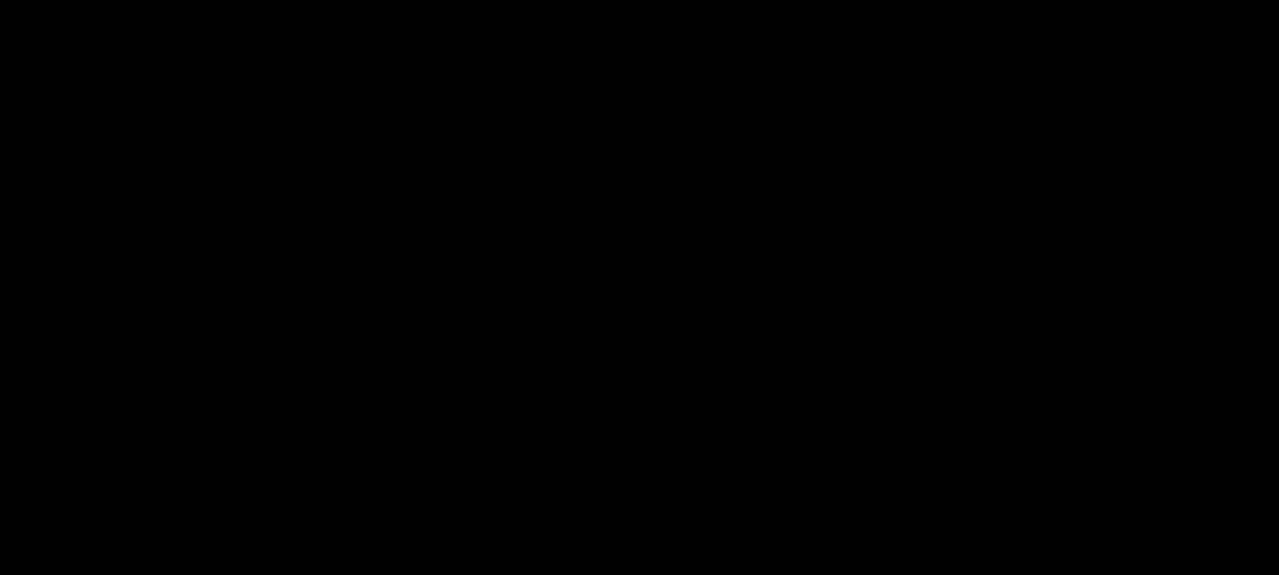
hasSolidBorder

Rectangle class and instances

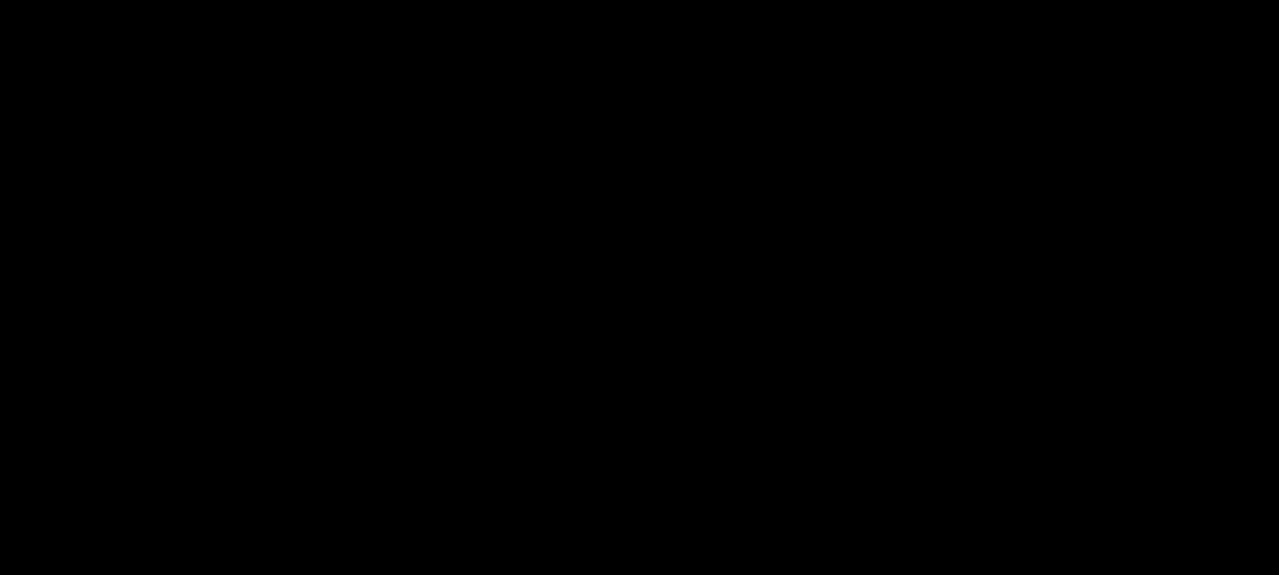
Class

```
public class Rectangle {  
    int width;  
    int height;  
    boolean hasSolidBorder;  
}
```

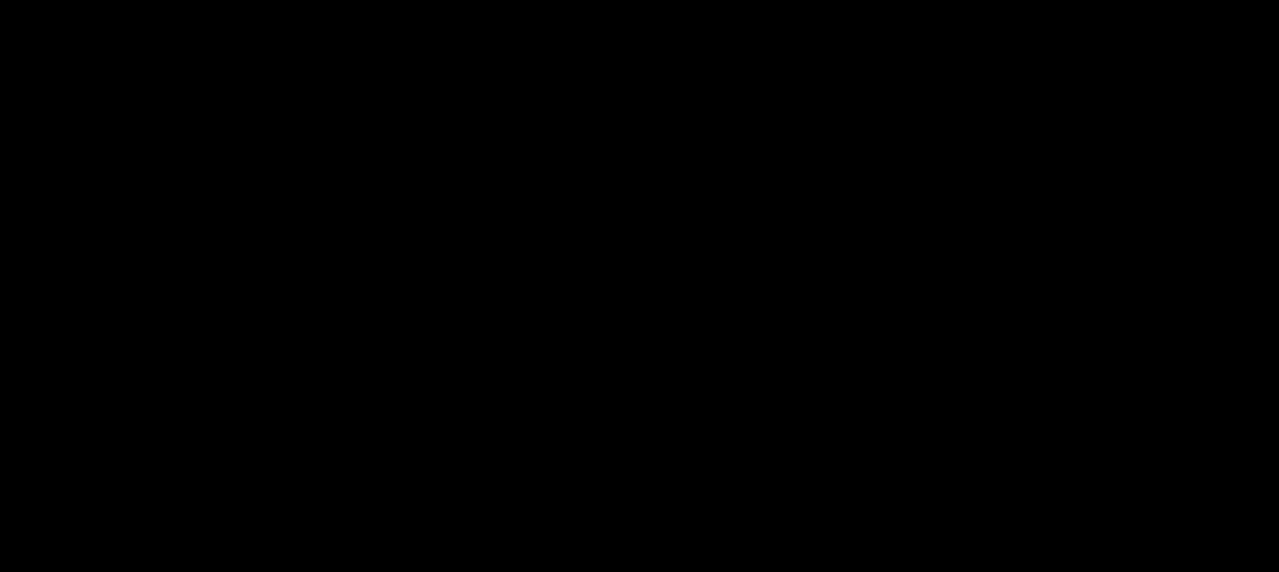
Rectangle class and instances



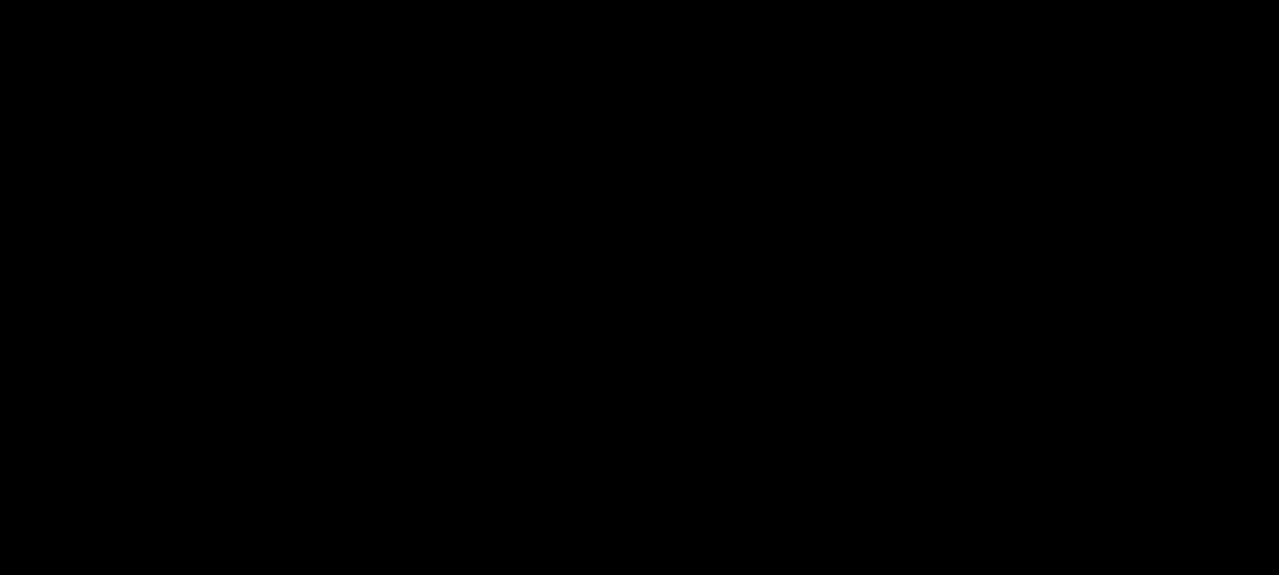
Rectangle class and instances



Rectangle class and instances



Rectangle class and instances



Generated diagrams

Rectangle

width

height

hasSolidBorder

Rectangle

f width int

f height int

f hasSolidBorder boolean

The new operator: Creating rectangle instances

```
Rectangle dashedRectangle = new Rectangle();
```

```
Rectangle solidRectangle = new Rectangle();
```

```
...
```

Syntax creating instances

`new class-name ([argument 1[, argument 2] ...])`

Wording examples:

- “Create an instance of class Rectangle”.
- “Create a Rectangle object.”
- “Create a Rectangle”.

Assigning attribute values to class instances

```
Rectangle dashedRectangle = new Rectangle();
```

```
dashedRectangle.width = 28;
```

```
dashedRectangle.height = 10;
```

```
dashedRectangle.hasSolidBorder = false;
```

Syntax accessing object attributes:

```
variable.attributeName = value;
```


References and null

```
Rectangle r = new Rectangle(); // Creating an object
```

```
r.width = 28; // o.K
```

```
r = null; // removing reference to object
```

```
r.width = 28; // Runtime error: NullPointerException (NPE)
```

```
Exception in thread "main" java.lang.NullPointerException  
at de.hdinstuttgart.mi.classdemo.App.main(App.java:17)
```


Checking for object presence

```
Rectangle r;
```

```
... // possible object assignment to variable r.
```

```
if (null == r) {  
    System.out.println("No rectangle on offer");  
} else {  
    System.out.println("Width: " + r.width);  
}
```

Why packages ?

- Grouping of related classes (e.g. subsystems).
- Structuring big systems.
- Provide access restrictions using:
publ i c, pri vat e and prot ect ed modifier
- Resolving class name clashes. Example:

java.lang.String vs. **my.personal**.St r i ng

Rules and conventions

- Package names below **java.** are reserved.
- Package names should not start with **javax.** either.
- Package names must not contain operators:
`mi . hdm st ut t gar t . de --> de.hdm_stuttgart.mi.`
- Packages should start with reversed DNS avoiding clashes.

Fully qualified class name vs. import

Fully qualified class name:

```
java.util.Scanner ❶ scanner = // Clumsy and  
    new java.util.Scanner ❷(System.in); // redundant
```

Using import:

```
import java.util.Scanner; ❶  
  
public class Q {  
  
    public static void main(String[] args) {  
        Scanner ❷ scanner = new Scanner ❷(System.in);  
        ...  
    }  
}
```

Don't be too lazy!

Bad

```
import java.util.*;

public class Q {
    public static void
        main(String[] args) {
        Scanner s =
            new Scanner(System.in);
        Date today = new Date();
    }
}
```

Good

```
import java.util.Scanner;
import java.util.Date;

public class Q {
    public static void
        main(String[] args) {
        Scanner s =
            new Scanner(System.in);
        Date today = new Date();
    }
}
```

Special: Classes in package java.lang

```
import java.lang.String; ❶ // Optional
import java.util.Scanner; ❷ // Required
public class Q {

    public static void main(String[] args) {
        String message = "Hello!";
        Scanner ❸ s = new Scanner(System.in);
    }
}
```

Class, package and file system

The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The breadcrumb navigation at the top reads: my > first > javapackage > Print. The Project view on the left shows a tree structure: target > classes > my > first > javapackage > Print.class. A small circle with the number '2' is next to Print.class. The code editor shows the file Print.java with the following code:

```
1 package my.first.javapackage
2
3 public class Print {
4     public void print(int i)
5         System.out.println("
6     }
7     public void print()
```

A small circle with the number '1' is next to the package declaration on line 1. The code is color-coded: package is blue, my.first.javapackage is grey, public class is blue, Print is blue, { is blue, public void is blue, print is blue, (int i) is blue, System.out.println is purple, " is blue, } is blue, and public void print() is blue.

Source hierarchy view

The image shows a file explorer interface with a source hierarchy view. The left sidebar contains navigation options: Recent, Home, Desktop, Trash, klausur, Documents, Music, and GoikLectures. The main area displays a tree structure of folders and files, with the following items highlighted by red boxes and numbered 1 through 5:

- 1. `java` folder
- 2. `my` folder
- 3. `first` folder
- 4. `javapackage` folder
- 5. `Print.java` file

The right pane shows the code content of `Print.java`:

```
package my.first.javapackage;

public class Print {
    ...
}
```

Name	Size	Type
<code>...</code>	2 items	Folder
<code>...</code>	1 item	Folder
<code>...</code>	1 item	Folder
<code>...</code>	1 item	Folder
<code>Print.java</code>	413 bytes	Text

Object methods

Change an object's state.

Example: Scale a rectangle.

Get dependent values

Example: Calculate a rectangle's perimeter.

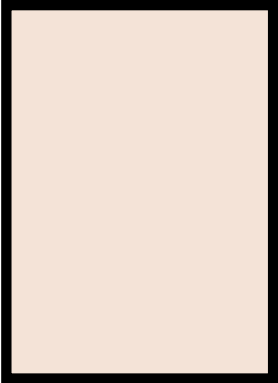
Combined

Scale a rectangle and calculate its new perimeter.

Scaling a rectangle

width=20

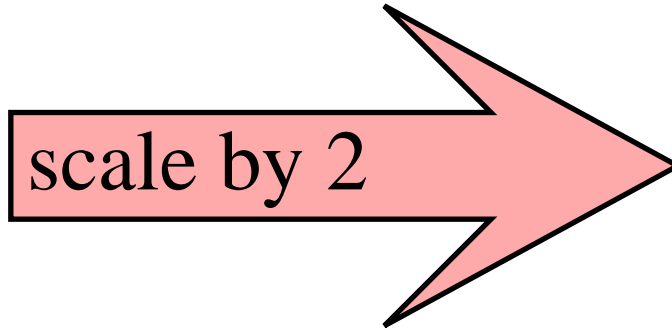
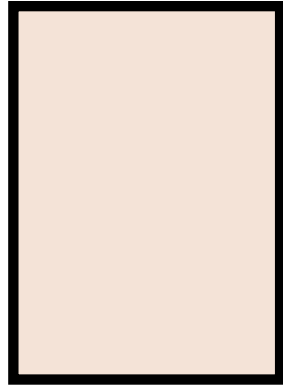
height
=30



Scaling a rectangle

width=20

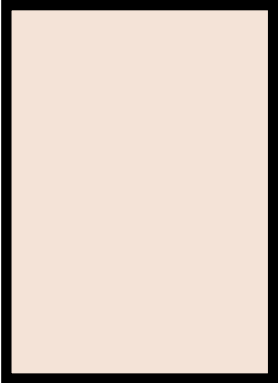
height
=30



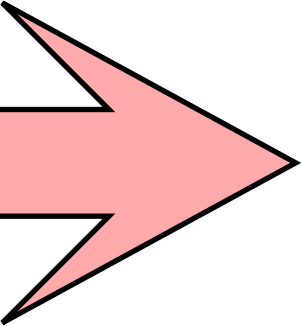
Scaling a rectangle

width=20

height
=30










scale by 2



Scaling method implementation

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    public void scale (int factor) {  
        width *= factor;  
        height *= factor;  
    }  
}
```

  Rectangle	
 width	int
 height	int
 hasSolidBorder	boolean
  scale(int)	void

Scaling method signature

```
void ❶ scale (int factor ❷) {  
...}
```

❶ No value is being returned to caller.

❷ A single value of type `int` is being provided as method argument.

Using the `scale(...)` method

```
Rectangle r = new Rectangle();  
r.width = 33;  
r.height = 22;
```

```
r.scale(2);
```

```
System.out.println("width=" + r.width);  
System.out.println("height=" + r.height);
```

```
width=66  
height=44
```

Method definition syntax

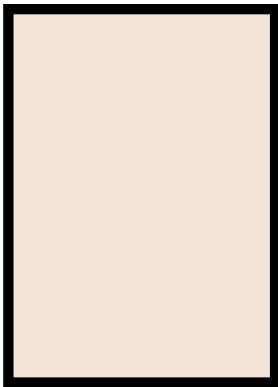
```
public ❶ void ❷ scale❸ (int factor ❹) { ❺  
    width *= factor; ❻  
    height *= factor;  
}
```

```
[access modifier] ❶ return_type ❷ methodName ❸ ([arguments] ❹) { ❺  
    [statement(s)] ❻  
}
```


A rectangle's perimeter

width=20

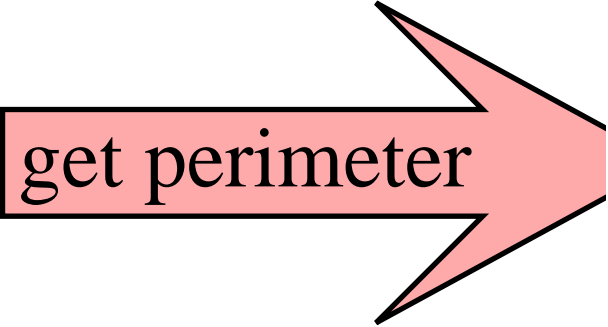
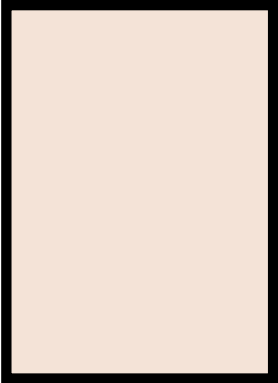
height
=30



A rectangle's perimeter

width=20

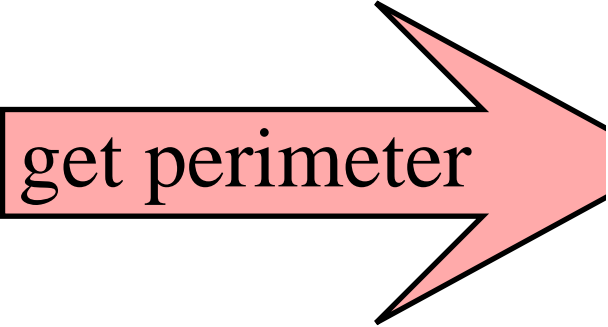
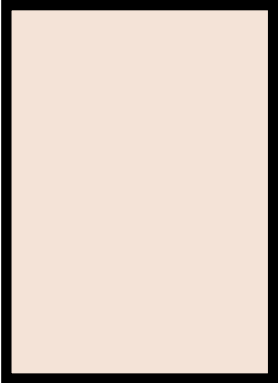
height
=30



A rectangle's perimeter










width=20

height
=30



getPerimeter() method implementation

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    public void scale (int factor) { ... }  
  
    public int getPerimeter() {  
        return 2 * (width + height);  
    }  
}
```

  Rectangle	
 width	int
 height	int
 hasSolidBorder	boolean
  scale(int)	void
  getPerimeter()	int

Using Rectangle.getPerimeter()

```
Rectangle r = new Rectangle();
```

```
r.width = 33;
```

```
r.height = 22;
```

```
System.out.println("Perimeter=" + r.getPerimeter());
```

```
Perimeter=110
```

Related exercises

Exercise 83: Compile time error

Exercise 84: An Address class

Access control: Overall objectives

- Fine-grained control on attributes and methods.
- Support encapsulation / Information hiding.

Purpose: Hide implementation details within class or package

Example: Two ways implementing a day's time

```
public class DayTime {  
    private int ❶ minutes_since_0_00;  
  
    public int getMinute() { ❷  
        return minutes_since_0_00 % 60;  
    }  
    public int getHour() { ❷  
        return minutes_since_0_00 / 60;  
    }  
}
```

```
public class DayTime {  
    private int minute, hour; ❶  
  
    public int getMinute() { ❷  
        return minute;  
    }  
    public int getHour() { ❷  
        return hour;  
    }  
}
```


Access violation

No access to private field of alien class Time:

```
public class Q {  
    public static void main(String[] args) {  
  
        Time t = new Time();  
  
        // Error: 'minutes_since_0_00' has private access in 'Time'  
        t.minutes_since_0_00 = 371;  
    }  
}
```

Access rules

Access Level	Other package	Child class	Same package	Same class
public	yes	yes	yes	yes
protected	no	yes	yes	yes
Default	no	no	yes	yes
private	no	no	no	yes

“Tips on Choosing an Access Level”

- Use the most restrictive access level that makes sense for a particular member.
- Use `private` unless you have a good reason not to.
- Avoid `public` fields except for constants. Public fields tend linking to a particular implementation and limit your flexibility in changing your code.

Related exercises

Exercise 85: Understanding access control

Exercise 86: Explaining times

Direct access vs. setter method

Direct access

```
public class Time {  
    public int hour, minute;  
}
```

```
Time time = new Time();
```

```
time.hour = 17;
```

```
time.minute = 45;
```

Setter access

```
public class Time {  
    private int hour, minute;  
    public void setTime(int h, int m) {  
        minute = m;  
        hour = h;  
    }  
}
```

```
Time time = new Time();
```

```
time.setTime(17, 45);
```

Why adding setter methods?

- Allow for change of implementation.
- Allow for non-business logic concerns. Examples:
 - Logging
 - Adding persistence (Databases)

```
public class Time {
    private int hour, minute;
    public void setTime(int h, int m) {
        minute = m;
        hour = h;
        System.out.println("Time has been set to "
            + hour + ":" + minute);
    }
}
```

Implementation change: Minutes only, no hours

Direct access

```
public class Time {
    // Minutes since 00:00
    public int minute;
}

final Time time = new Time();
time.minute = 17 * 60 + 45;
```

Setter / getter access

```
public class Time {
    private int minute; // Minutes since 00:00
    public void set(int h, int m) {
        minute = m + 60 * h;
    }
    public int getMinute() { /* coming soon */ }
    public int getHour() { /* coming soon */ }
}

final Time time = new Time();
time.setTime(17, 45);
```

Related exercises

Exercise 87: Implementing getter methods

Defining type signatures

`boolean` ❶ `startsWith(String prefix, int offset)` ❷

- ❶ Return type `boolean`
- ❷ Arguments among with their respective types and order:
 1. Type `String`
 2. Type `int`

Type signature examples

Method	Return type	Argument type list
<code>void print()</code>	<code>void</code>	<code>(void)</code>
<code>int add (int a, int b)</code>	<code>int</code>	<code>(int, int)</code>
<code>int max (int a, int b)</code>	<code>int</code>	<code>(int, int)</code>
<code>void print (int a, float b)</code>	<code>void</code>	<code>(int, float)</code>
<code>void display (float a, int b)</code>	<code>void</code>	<code>(float, int)</code>

Defining method signatures

`boolean startsWith①(String prefix, int toffset) ②`

- ① Method name `startsWith`.
- ② Number of arguments along with their respective types and order.

Method signature examples

Method	Method name	Method signature
<code>void print()</code>	<code>print</code>	<code>(void)</code>
<code>int add (int a, int b)</code>	<code>add</code>	<code>(int, int)</code>
<code>int max (int a, int b)</code>	<code>max</code>	<code>(int, int)</code>
<code>void print (int a, float b)</code>	<code>print</code>	<code>(int, float)</code>
<code>void display(float a, int b)</code>	<code>display</code>	<code>(float, int)</code>

Related exercises

Exercise 88: Method signature variants

Method overloading: Same name, different signature

```
public class Print {  
    public void print() { // (void)  
        System.out.println("No argument");  
    }  
    public void print(int i) { // (int)  
        System.out.println("int value " + i);  
    }  
    public void print(double d) { // (double)  
        System.out.println("double value " + d);  
    }  
    public void print(int i, int j) { // (int, int)  
        System.out.println("Two int values " +  
            i + " and " + j);  
    }  
}
```

```
Print p = new Print();  
p.print();  
p.print(33);  
p.print(4.333);  
p.print(-1, 7);  
No argument  
int value 33  
double value 4.333  
Two int values -1 and 7
```

Overloading, alternate names

- Static polymorphism.
- Compile time binding.
- Early binding.

No overloading in »C«

```
include <stdio.h>
```

```
void print() {  
    printf("No argument\n");  
}
```

```
void print(int i) { /* Error: redefinition of 'print' */  
    printf("int value %d\n", i);  
}
```

```
void main(void) {  
    print();  
    print(33);  
}
```


»C« requires unique function names

Code in file `print.c`

```
include <stdio.h>

void print() {
    printf("No argument\n");
}

/* Different function name */
void printIntValue(int i) {
    printf("int value %d\n", i);
}

void main(void) {
    print();
    printIntValue(33);
}
```

Compile / execute

Compiling `print.c` to executable file `print`

```
> cc -o print print.c
```

Executing file `print`

```
> ./print
No argument
int value 33
```

No distinction on return type

```
public class Person {  
    String getDetails() { return "dummy";}  
    int getDetails() { return 1;} // Error: 'getDetails()' is already  
}                               // defined in 'Person'
```

Return type	Method signature	
	Method name	Argument type list
String	getDetails	(void)
int	getDetails	(void)

Only method signature support in Java™ ignoring return type.

Method signatures rationale

In Java™ method signatures allow for uniquely addressing a method within a given class e.g.:

The method named `print` having an `int` argument followed by a `double`:

```
print (int, double)
```

Method signatures rationale

```
Print p = new Print();  
p.print(3.14);
```

```
public class Print {  
    public void print(int i) {...}  
    public void print() {...}
```

Related exercises

Exercise 89: Will a match be found?

Example: System.out.print(...)

`print(boolean b)`

`print(char[] s)`

`print(float f)`

`print(long l)`

`print(String s)`

`print(char c)`

`print(double d)`

`print(int i)`

`print(Object obj)`

Creating and initializing rectangles

<pre>int a; a = 33;</pre>	<pre>Rectangle r = new Rectangle(); r.width = 28; r.height = 10; r.hasSolidBorder = false;</pre>
-------------------------------	---

Combining statements desired:

<pre>int a = 33; // works!</pre>	<pre>Rectangle r = new Rectangle(28, 10, false); //how ???</pre>
----------------------------------	--

Defining a constructor

```
public class Rectangle {
    int width, height;
    boolean hasSolidBorder;

    ...
    public ❶ Rectangle ❷ (int width, int height, boolean hasSolidBorder) {
        this.width = width;
        this.height = height;
        this.hasSolidBorder = hasSolidBorder;
    }
}
```


Constructor syntax

```
[access modifier] constructorName (listOfArguments) {  
    [constructor body]  
}
```

Empty argument list Default constructor e.g. `obj = new MyClass()` .

Non-empty argument list Non-default constructor e.g. :
`obj = new String("xyz");`

Constructors











- Can only be executed on object creation.
- Are being called prior to any non-constructor method.
- Only one of potentially multiple constructors will be executed exactly one time.

However nesting is possible.

Multiple overloaded constructors

```
public class Rectangle {
    int width, height;

    public Rectangle() {
        width = height = 1;
    }
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle(int widthAndHeight) {
        width = height = widthAndHeight;
    }
}
```

  Rectangle	
 width	int
 height	int
  Rectangle()	
  Rectangle(int, int)	
  Rectangle(int)	

Constructor calls within constructor

```
public class Rectangle {
    int width, height;











    public Rectangle(int width,
                    int height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle() {
        width = height = 1;
    }
    public Rectangle(
        int widthAndHeight) {
        width = height =
            widthAndHeight;
    }
}
```

```
public class Rectangle {
    int width, height;

    public Rectangle(int width,
                    int height) {
        this.width = width;
        this.height = height;
    }
    public Rectangle() {
        this(1, 1); ❶
    }
    public Rectangle(
        int widthAndHeight) {
        this(widthAndHeight, ❷
            widthAndHeight);
    }
}
```

Instances by overloaded constructors

```
Rectangle standard = new Rectangle(); // 1 x 1  
Rectangle square = new Rectangle(2); // 2 x 2  
Rectangle individual = new Rectangle(2, 7); // 2 x 7
```

  Rectangle	
 width	int
 height	int
  Rectangle()	
  Rectangle(int, int)	
  Rectangle(int)	

No constructor vs. default constructor

Equivalent: `Rectangle r = new Rectangle();`

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    // Default constructor, empty body.  
    public Rectangle ( ) {}  
}
```

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    // No constructor at all  
}
```

Non - default constructor, but no default constructor

```
public class Rectangle {
    int width, height;
    boolean hasSolidBorder;

    // Non-default constructor
    public Rectangle(int width,
                    int height,
                    boolean hasSolidBorder) {
        this.width = width;
        this.height = height;
        this.hasSolidBorder =
            hasSolidBorder;
    }

    // No defined default constructor.
}
```

```
// o. K.: Using non-default
// constructor.
```

```
Rectangle r =
    new Rectangle(3, 6, false);
```

```
// Wrong: Default constructor
// undefined, but non-default
// constructor present.
```

```
Rectangle r = new Rectangle();
```

Related exercises

Exercise 90: Modeling geometry objects: Rectangles

Exercise 91: Modeling circles

Exercise 92: Adding translations and SVG export.

Exercise 93: Extending the employee example.

Exercise 94: Refining access to an employee's attributes

Exercise 95: File system representation

Exercise 96: Your personal `String` class

Circle and variable scopes

```
public class Circle {  
    private double radius;  
  
    public Circle(double r){  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Circle and variable scopes

```
public class Circle {  
    private double radius;  
  
    public Circle(double r){  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Instance variable »**radius**« is visible
on class level including all methods

Circle and variable scopes

```
public class Circle {  
    private double radius;  
  
    public Circle(double r){  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Scope of constructor argument variable »**r**« is limited to constructor

Documenting classes and methods

```
/** Representing circles.
 */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param r representing the circle's radius
     */
    public Circle(double r) {
        radius = r;
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

Constructor Detail

Circle

```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius

Generated Javadoc

Constructor Detail


Circle

```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius



Bad: Choosing variable name
»r« rather than self-explanatory
»radius«

Generated Javadoc

Constructor Detail

Circle


```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius

Bad: Choosing variable name
»r« rather than self-explanatory
»radius«



Choice requires supplementary explanation



Generated Javadoc

Constructor Detail

Circle

```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius

Generated Javadoc

Constructor Detail

Circle **Non-self-explanatory**
variable name »r«

```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius

Constructor Detail

Circle **Self-explanatory**
variable name »radius«

```
public Circle(double radius)
```

Creating a circle.

Parameters:

radius - Circle's size

Refactoring «r» «radius»

```
/** Representing circles. */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param radius Circle's size
     */
    public Circle(double radius) {
        radius = radius;
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

Refactoring «r» «radius»

```
/** Representing circles. */  
public class Circle {  
    private double radius;  
  
    /** Creating a circle.  
     * @param radius Circle's size  
     */  
    public Circle(double radius) {  
        radius = radius;  
    }  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Constructor Detail

Circle

```
public Circle(double radius)
```

Creating a circle.

Parameters:

radius - Circle's size



Refactoring «r» «radius»

```
/** Representing circles. */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param radius Circle's size
     */
    public Circle(double radius) {
        radius = radius;
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

Constructor Detail

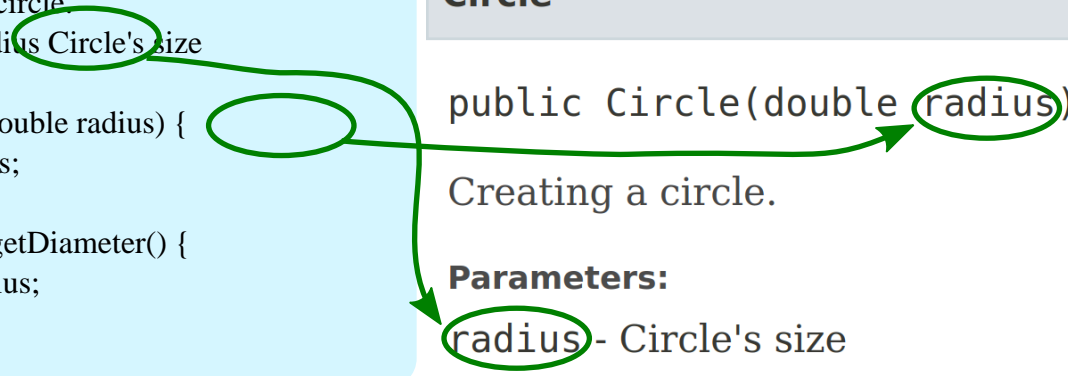
Circle

public Circle(double radius)

Creating a circle.

Parameters:

radius - Circle's size



Scope assignment problem

```
/** Representing circles.
 */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param radius Circle's size
     */
    public Circle(double radius) {
        radius = radius; // Warning: Variable 'radius' is assigned to itself.
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

this overriding method scope

```
public class Circle {  
    private double radius;  
  
    public Circle(double r) {  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

this overriding method scope


```
public class Circle {  
    private double radius;  
  
    public Circle(double r) {  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Refactor »r« to »radius«

this overriding method scope

```
public class Circle {  
    private double radius;  
  
    public Circle(double radius){  
        radius = radius;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```


Scope Problem: Self-assignment
rather than intended assignment
to instance variable **radius**



this overriding method scope

```
public class Circle {  
    private double radius;  
  
    public Circle(double radius){  
        radius = radius;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```


Solution: Resolve scope conflict
by qualifying desired instance
variable »**radius**«



this overriding method scope

```
public class Circle {  
    private double radius;  
  
    public Circle(double radius){  
        this.radius = radius;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

The »**this**« keyword relates to **instance scope** resolving method scoped **radius** from instance scoped radius.



Related exercises

Exercise 97: Constructors variable names and “this”.

Why do we require an instance?

```
public class Helper {  
    // Find the larger of two values  
    public int maximum(int a, int b) {  
        if (a < b) {  
            return b;  
        } else {  
            return a;  
        }  
    }  
}
```

```
final Helper instance = new Helper();  
  
// Why do we need an instance just for  
// computing the maximum of two values?  
System.out.println("Maximum " +  
    instance.maximum(-3, 5));  
  
Maximum 5
```

Observation: The instance's state is irrelevant for finding the maximum of two values.

Solution: Replace instance method by class method using **static**

```
public class Helper { ❶  
    static ❷ public int maximum(int a, int b)  
    {  
        if (a < b) {  
            return b;  
        } else {  
            return a;  
        }  
    }  
}
```

```
// No instance required any longer  
System.out.println("Maximum " +  
    Helper.maximum(-3, 5)); ❸
```

```
Maximum 5
```

Club membership objectives

- Each club member has got a name.
- Each member has got an ascending unique membership number.
- The overall club's member count needs to be accounted for.

Solution:

- **Class level:** Advance club's member count by each new member.
- **Instance level:** New members receive name and current member count plus 1.

Step 1: Implementing club member names.

```
public class ClubMember {  
  
    final private String name;  
  
    public ClubMember(final String name) {  
        this.name = name;  
    }  
    public String toString() {  
        return "Member " + name;  
    }  
}
```

Showing membership info.

```
final ClubMember
```

```
john = new ClubMember("John"),  
karen = new ClubMember("Karen");
```

```
System.out.println(john.toString());  
System.out.println(karen.toString());
```

```
Member John
```

```
Member Karen
```


Step 2: Adding membership numbers.

```
public class ClubMember {  
  
    static ❶ private int memberCount = 0;  
  
    final private int memberNumber; ❷  
    final private String name;  
  
    public ClubMember(final String name) {  
        this.name = name;  
        memberNumber = ++memberCount; ❸  
    }  
    public String toString() {  
        return "Member " + name + ", member number " + memberNumber ❹;  
    }  
}
```

Showing membership numbers.

```
final ClubMember
```

```
    john = new ClubMember("John"),  
    karen = new ClubMember("Karen");
```

```
System.out.println(john); ❶ // toString() is being  
System.out.println(karen); // called implicitly
```

Member John, member number 1

Member Karen, member number 2

Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```

Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```



Remark: Happens prior to creating any instance of ClubMember.

Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```

0

```
new ClubMember("John")
```

**Remark: Creating first
Instance of class
ClubMember.**

Member creation steps

```
public class ClubMember {
```

0

```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

```
    public ClubMember(
```

```
        final String name) {
```

```
        this.name = name;
```

```
        memberNumber = ++memberCount;
```

```
    }
```

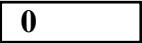
```
}
```

```
new ClubMember("John")
```



Member creation steps

```
public class ClubMember {
```



```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

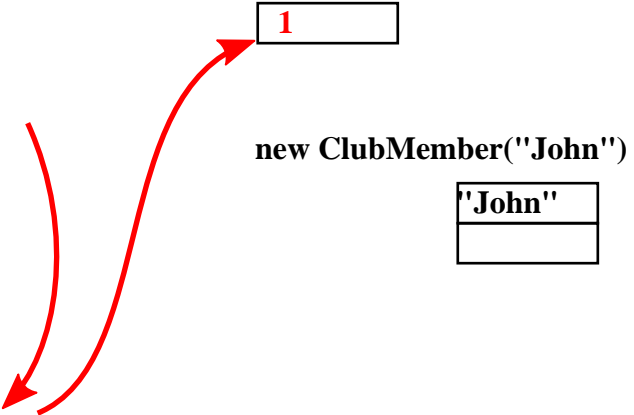
```
new ClubMember("John")
```



```
    public ClubMember(
        final String name) {
        this.name = name;
        memberNumber = ++memberCount;
    }
}
```

Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```

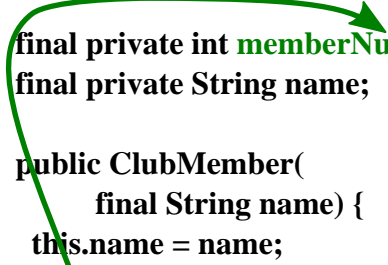


Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```

1

new ClubMember("John")



Member creation steps

```
public class ClubMember {
```

1

```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

```
    public ClubMember(
```

```
        final String name) {
```

```
        this.name = name;
```

```
        memberNumber = ++memberCount;
```

```
    }
```

```
}
```

```
new ClubMember("John")
```

'John'

1

```
new ClubMember("Karen")
```

Member creation steps

```
public class ClubMember {
```

1

```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

```
    public ClubMember(
```

```
        final String name) {
```

```
        this.name = name;
```

```
        memberNumber = ++memberCount;
```

```
    }
```

```
}
```

```
new ClubMember("John")
```

'John'

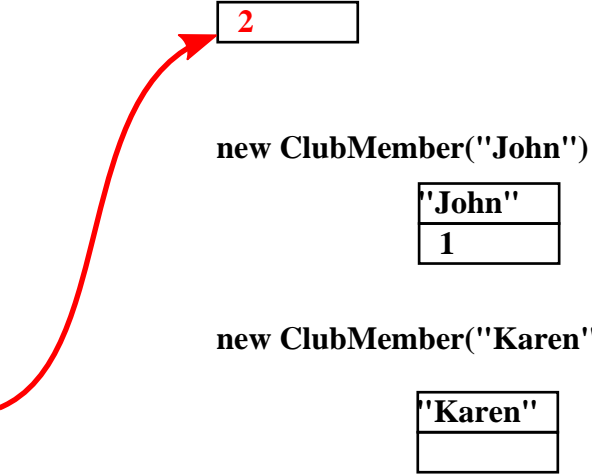
1

```
new ClubMember("Karen")
```

'Karen'

Member creation steps

```
public class ClubMember {  
  
    static private int memberCount = 0;  
  
    final private int memberNumber;  
    final private String name;  
  
    public ClubMember(  
        final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
  
}
```



Member creation steps

```
public class ClubMember {
```

2

```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

```
    public ClubMember(  
        final String name) {
```

```
        this.name = name;
```

```
        memberNumber = ++memberCount;
```

```
    }
```

```
}
```

```
new ClubMember("John")
```

"John"

1

```
new ClubMember("Karen")
```

"Karen"

2

Accessing the club's overall member count?

```
public class ClubMember {  
  
    static private int memberCount = 0;  
    ...  
    public ClubMember(final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
    static public int getMemberCount() ❶ {  
        return memberCount; ❷  
    }  
    ...  
}
```

Accessing the club's member count

```
final ClubMember  
    john = new ClubMember("John"),  
    karen = new ClubMember("Karen"),  
    petra = new ClubMember("Petra");
```

```
System.out.println(karen.toString());
```

```
System.out.println("Club's member count:"  
    + ClubMember.getMemberCount());  
// Good: Prevent tampering memberCount  
// variable.
```

```
Member Karen, member number 2  
Club's member count: 3
```

Syntax accessing class members

Class Variable { class name }. { variableName }

Class Method { class name }. { methodName } ([parameter])

static / non-static wrap up

```
public class X {  
    int a; ❶  
    static int b; ❷
```

- ❶ Variable a defined **once per instance** of class X
- ❷ Variable b defined **once per class X**.

Finally understanding `System.out.println()`

① ② ③

`System.out.println(...)`

- ① Class `System` in package `java.lang`.
- ② Class variable (static) `out` of type `PrintStream` in class `System`
- ③ One of 9 overloaded methods in class `PrintStream`

Related exercises

Exercise 98: Class vs. instance

Exercise 99: Distinguishing leap- and non-leap years

Exercise 100: A method for printing square numbers using `for`, `while` and `do ... while`

Exercise 101: Nicely formatting sine values.

Exercise 102: Extending our interest calculator

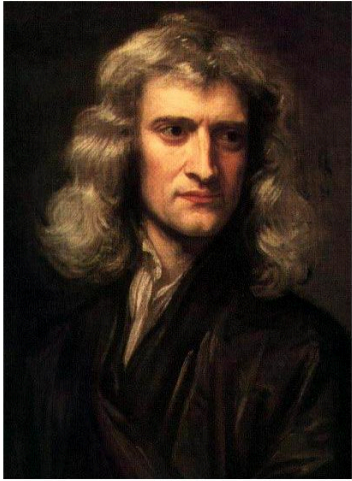
Exercise 103: Integer value considerations.

Exercise 104: Programmer's favourite expression

Exercise 105: Lotteries revisited

Exercise 106: Finding the greatest common divisor of two integer values

Newton's letter to Robert Hooke



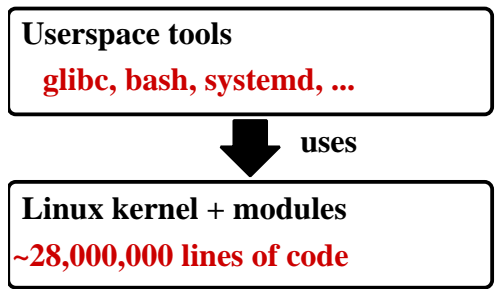
»If I have seen further
it is by standing on ye
sholders of Giants«



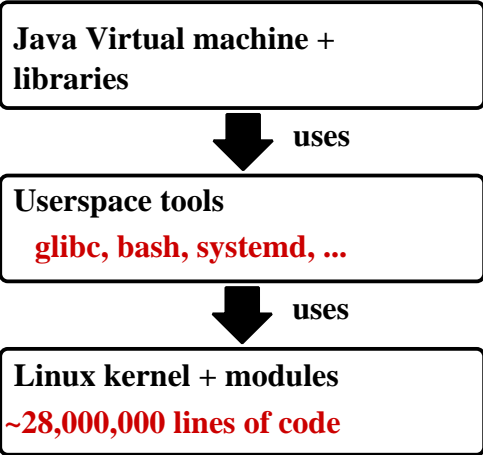
Application execution prerequisites

Linux kernel + modules
~28,000,000 lines of code

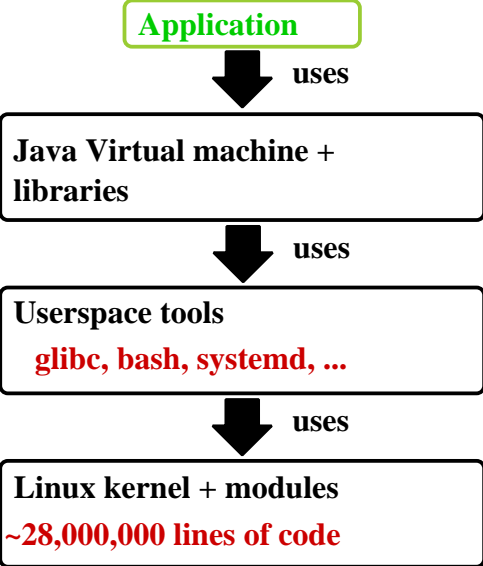
Application execution prerequisites



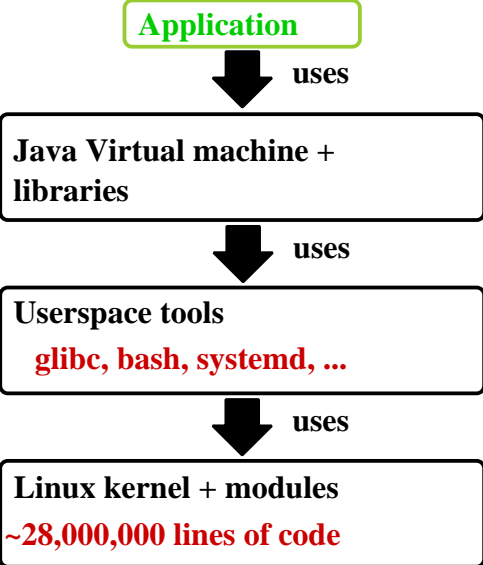
Application execution prerequisites



Application execution prerequisites

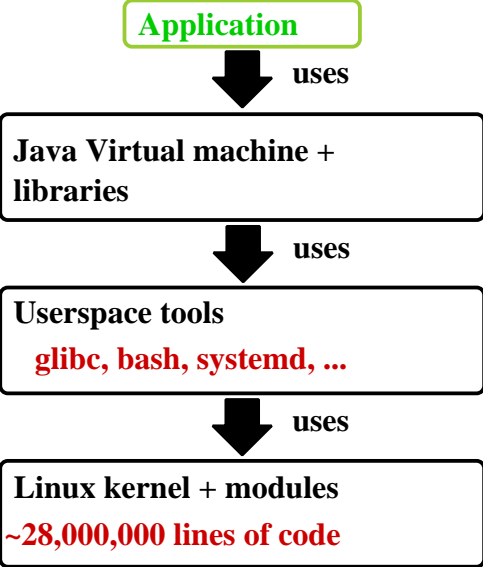


Application execution prerequisites



- Tools:**
- Compiler
 - Debugger
 - IDE
 - ...

Application execution prerequisites



- Tools:**
- Compiler
 - Debugger
 - IDE
 - ...

- Services:**
- Network
 - Databases
 - Web services
 - ...

Why Maven project management?

- Automated third-party class import and dependency management
- Executing automated tests
- Complete project lifecycle:
compile, test, execute, package, deploy
- Extensible plugin architecture

Example: Creating PDF using iText7

Source: CreatePdf.java

```
import com.itextpdf.kernel.pdf.PdfDocument;
...
final PdfWriter writer = new PdfWriter("helloWorld.pdf");

final PdfDocument pdf = new PdfDocument(writer);

final Document document =
    new Document(pdf, PageSize.A8.rotate());

document.add(new Paragraph("Hello World!").
    setFontColor(ColorConstants.MAGENTA).
    setFontSize(20));

document.close();
```

PDF output

Hello World!

Maven iText library pom.xml definition



iText Core » 8.0.3

A Free Java-PDF library

License	AGPL 3.0
Tags	pdf
Organization	Apryse Group NV
HomePage	https://itextpdf.com/
Date	Feb 07, 2024
Files	pom (4 KB) View All
Repositories	Central

```
<project ... >
...
<dependencies>
  <dependency>
    <groupId>com itext pdf </groupId>
    <artifactId>i text - core</artifactId>
    <version>8. 0. 3</version>
    <type>pom /type>
  </dependency>
</dependencies>
...
</project >
```

ltext transitive dependencies

mvn dependency: tree

...

--- dependency: 3.6.1: tree (default-cli) @ hellopdf ---

de.hdmstattgart.mi:hellopdf:jar:0.9

\- com.itextpdf:itext-core:pom:8.0.2:compile

+ com.itextpdf:barcodes:jar:8.0.2:compile

| \- org.slf4j:slf4j-api:jar:1.7.36:compile

+ com.itextpdf:font-asian:jar:8.0.2:compile

+ com.itextpdf:forms:jar:8.0.2:compile

+ com.itextpdf:hyph:jar:8.0.2:compile

+ com.itextpdf:io:jar:8.0.2:compile

| \- com.itextpdf:commons:jar:8.0.2:compile

+ com.itextpdf:kernel:jar:8.0.2:compile ...

Class location in iText library

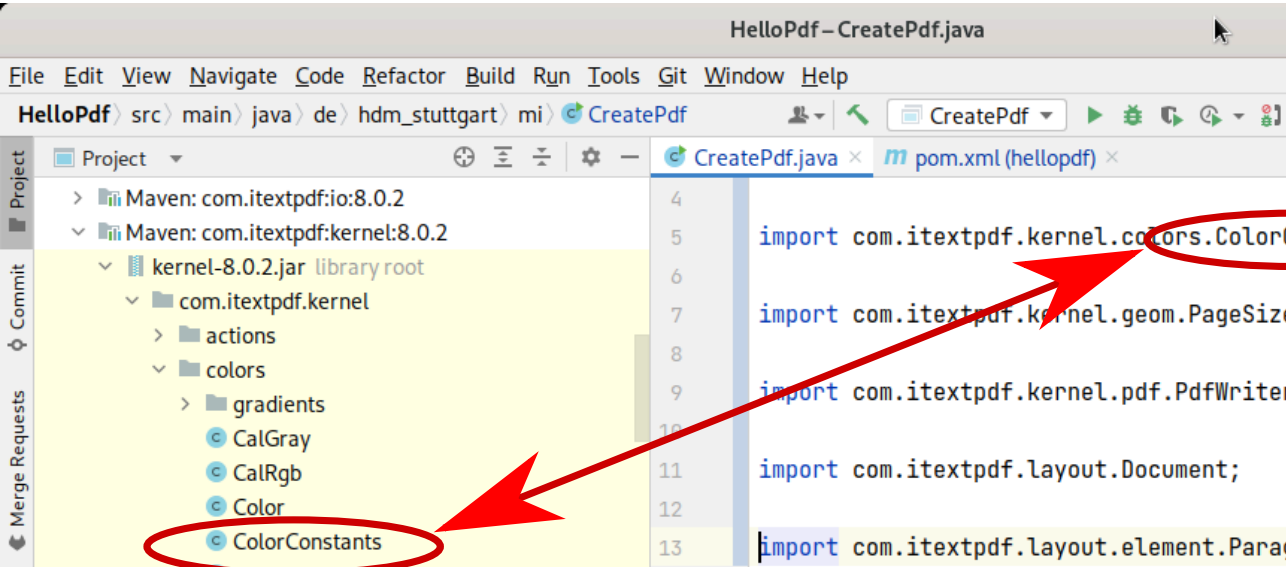
The screenshot shows an IDE window titled "HelloPdf - pom.xml (helloworld)". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, and Help. The breadcrumb shows "HelloPdf > m pom.xml". The toolbar contains icons for user, navigation, and a "CreatePdf" button. The left sidebar shows "Project" and "Commit" views. The "Project" view displays a tree of "External Libraries" with the following entries:

- > < 19 > /usr/lib/jvm/java-17-openjdk
- > Maven: com.itextpdf:barcodes:8.0.2
- > Maven: com.itextpdf:bouncy-castle-connector:8.0.2
- > Maven: com.itextpdf:commons:8.0.2
- > Maven: com.itextpdf:font-asian:8.0.2
- > Maven: com.itextpdf:forms:8.0.2
- > Maven: com.itextpdf:hyph:8.0.2
- > Maven: com.itextpdf:io:8.0.2
- > Maven: com.itextpdf:kernel:8.0.2
- > Maven: com.itextpdf:layout:8.0.2
- > Maven: com.itextpdf:pdfa:8.0.2

The main editor displays the XML content of the pom.xml file, with line numbers 19 through 28 on the left. The following XML snippet is visible, with the version "8.0.2" highlighted in blue:

```
19
20 <dependencies>
21   <dependency>
22     <groupId>com.itextpdf</groupId>
23     <artifactId>itext7-core</artifactId>
24     <version>8.0.2</version>
25     <type>pom</type>
26   </dependency>
27
28 </dependencies>
```

Class location in iText library



Maven repositories

CDN	Maven Central
Company local	Sonatype Nexus, e.g. MI Maven repository

Related exercises

Exercise 107: Dealing with IBAN numbers

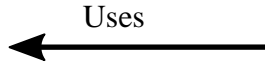
Maven archetypes

- Blueprints for projects.
- Based on the pom.xml file:
Project **O**bject **M**odel
- Initial project creation.
- CLI and IDE support.
- Extensibility by archetype catalogs.

```
mvn archetype:generate
[INFO] Scanning for projects...
...
1: remote -> amik.archetype:elmspring-boot-blank-archetype
2: remote -> amik.archetype:graalvmblank-archetype...
3: remote -> amik.archetype:graalvmspringmvc-blank-archetype
...
2083: remote -> org.apache.maven.archetypes:maven-archetype
...
3330: remote -> za.co.absa.hyperdrive:component-archetype
Choose a number or apply filter ...
```

Project «lottery» depending on «helper»

Project »helper«



Project »lottery«

```
public class Helper {  
    static public long  
    factorial(int n) {  
        long result = 1;  
        for (int i=2;i<=n;i++){  
            result *= i;  
        }  
        return result;  
    }  
}
```

```
a = Helper.factorial(6);
```

Providing project «helper»

Helper.java

```
package mi.calc.common;
public class Helper {
    static public long
        factorial(int n) {
        long result = 1;
        for (int i = 2;
            i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org/...>
    ...
    <groupId>mi.calc</groupId>
    <artifactId>common</artifactId>
    <version>1.0</version>
    ...
</project>
```

Install project «Common»

```
goik@goiki Helper> mvn install
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building helper 0.9
...
[INFO] Installing .../Helper/target/helper-0.9.jar to
     /na/goik/.m2/repository/mi/cal c/common/1.0/common-1.0.jar
```

hel per - 0.9.jar archive content

goi k@goi ki tmp> unzip ... hdm st ut t gart / de / mi / sd1 / hel per / 0.9 / hel per - 0.9.jar

Archive: ... /.n2/repository/... /sd1/hel per / 0.9 / hel per - 0.9.jar

creating: META-INF/

inflating: META-INF/MANIFEST.MF

creating: de/

creating: de/hdm st ut t gart /

creating: de/hdm st ut t gart / mi /

creating: de/hdm st ut t gart / mi / sd1 /

inflating: de/hdm st ut t gart / mi / sd1 / Hel per . class

creating: META-INF/naven/

creating: META-INF/naven/de.hdm st ut t gart . mi . sd1 /

creating: META-INF/naven/de.hdm st ut t gart . mi . sd1 / hel per /

inflating: META-INF/naven/de.hdm st ut t gart . mi . sd1 / hel per / pom.xml

inflating: META-INF/naven/de.hdm st ut t gart . mi . sd1 / hel per / pom.properties

Consuming project «Lottery»

```
<project ... >
```

```
...
```

```
<groupId>de. hdmst ut tgart. mi. sd1</groupId>
```

```
<artifactId>l ot tery</artifactId>
```

```
<version>0. 9</version>
```

```
<packaging>j ar</packaging>
```

```
<name>l ot tery</name>
```

```
<dependencies>
```

```
<dependency>
```

```
<groupId>de. hdmst ut tgart. mi. sd1</groupId>
```

```
<artifactId>hel per</artifactId>
```

```
<version>0. 9</version>
```

```
</dependency>
```

```
...
```

```
</project >
```


External libraries view

The screenshot shows the 'External Libraries' view on the left, which is expanded to show a Maven dependency. The dependency is for the artifact 'helper-0.9.jar' from the group 'de.hdm_stuttgart.de.mi.sd1'. The version is '0.9'. The code snippet on the right shows the XML representation of this dependency in a pom.xml file.

```
21  
22 <dependency>  
23   <groupId>de.hdm_stuttgart.de.mi.sd1</  
24   <artifactId>helper</artifactId>  
25   <version>0.9</version>  
26 </dependency>  
27
```

Using Helper.factorial(...) computing

```
static public long binomial(int n, int k) {
    return (Helper.factorial(n) / Helper.factorial(k)
           / Helper.factorial(n - k));
}

public static void main(String[] args) {
    System.out.println("There are " + binomial(5, 2) +
        " ways to draw 2 out of 5 numbers");

    System.out.println("There are " + binomial(49, 6) +
        " ways to draw 6 out of 49 numbers");
}
```

Maven artifact dependency.

```
public class Helper {  
    static public long factorial(int n) {  
        ... return result;  
    }  
}
```

Maven artifact dependency.

```
public class Helper {  
    static public long factorial(int n) {  
        ... return result;  
    }  
}
```

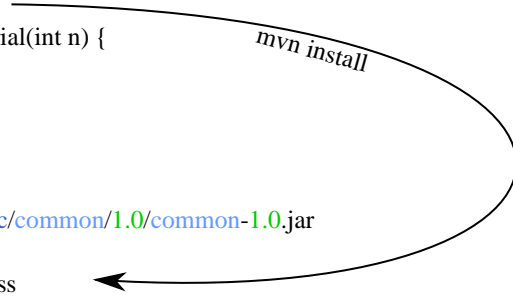
```
<project ...>...  
  <groupId>mi.calc</groupId>  
  <artifactId>common</artifactId>  
  <version>1.0</version> ...  
</project>
```

Maven artifact dependency.

```
public class Helper {  
    static public long factorial(int n) {  
        ... return result;  
    }  
}
```

```
jar tf ~/.m2/repository/mi/calc/common/1.0/common-1.0.jar  
...  
mi/calc/common/Helper.class
```

```
<project ...>...  
  <groupId>mi.calc</groupId>  
  <artifactId>common</artifactId>  
  <version>1.0</version> ...  
</project>
```



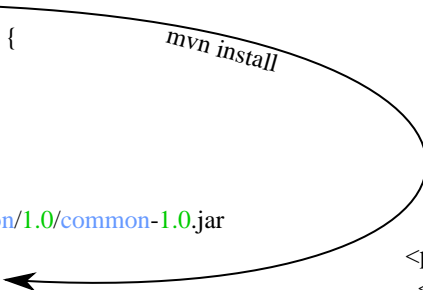
Maven artifact dependency.

```
public class Helper {  
    static public long factorial(int n) {  
        ... return result;  
    }  
}
```

```
jar tf ~/.m2/repository/mi/calc/common/1.0/common-1.0.jar  
...  
mi/calc/common/Helper.class
```

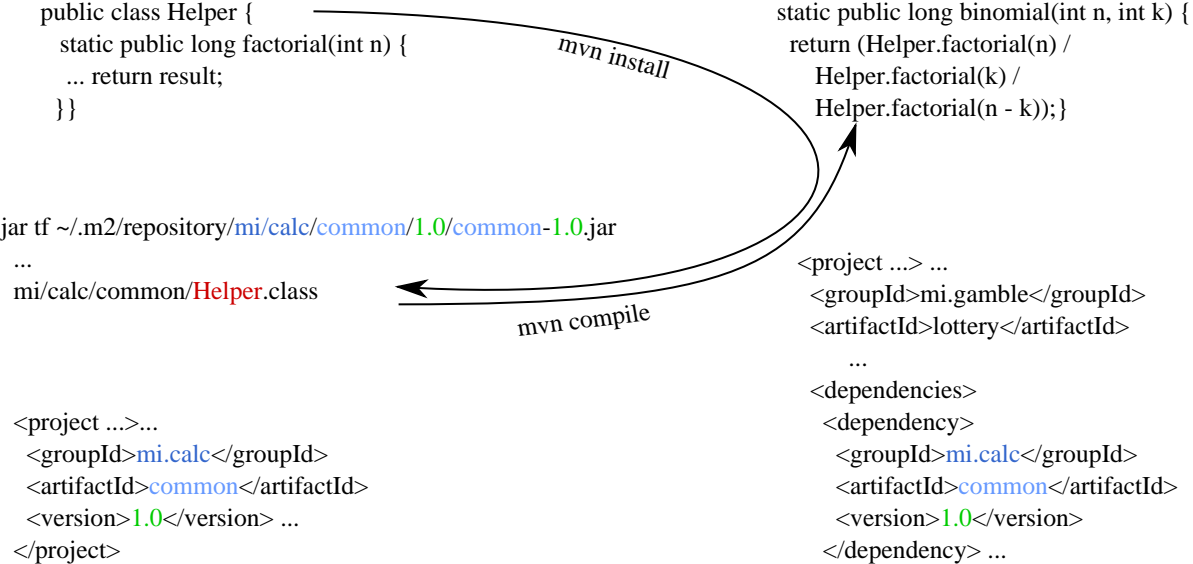
```
<project ...>...  
  <groupId>mi.calc</groupId>  
  <artifactId>common</artifactId>  
  <version>1.0</version> ...  
</project>
```

mvn install



```
<project ...> ...  
  <groupId>mi.gamble</groupId>  
  <artifactId>lottery</artifactId>  
  ...  
  <dependencies>  
    <dependency>  
      <groupId>mi.calc</groupId>  
      <artifactId>common</artifactId>  
      <version>1.0</version>  
    </dependency> ...  
  </dependencies>  
</project>
```

Maven artifact dependency.



Related exercises

Exercise 108: Cancelling fractions

Exercise 109: Dealing with local Maven dependencies

Using the MI Sd1 project template

- Download file `navenTempl at e. zi p` from `her e`.
- Extract `navenTempl at e. zi p` to folder `t empl at e`.
- Optional: Edit `t empl at e/pom.xml` reflecting your project needs i.e. `<groupI d>` and related.
- Optional: Import your project in IntelliJ IDEA.

CLI example

```
mvn --batch-mode -e archetype:generate \
-DgroupId=de.hdmitstuttgart.misdl -DartifactId=second -Dversion=0.9 \
-DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.0
```

```
[INFO] Scanning for projects...
```

```
...
[INFO] BUILD SUCCESS ...
```

See artifact reference.

Supplementary MI Maven archetypes

- MI nexus repository server
- Configuration:

```
mkdir -p ~/.m2 ❶
```

```
nano ~/.m2/settings.xml ❷
```

CLI testing mi-maven-archetype-quickstart

```
mvn --batch-mode -e archetype:generate \
  -DgroupId=de.hdm_stuttgart.mi.sd1 -DartifactId=second -Dversion=0.9 \
  -DarchetypeGroupId=de.hdm_stuttgart.mi -DarchetypeArtifactId=mi-maven-archetype-quickstart -DarchetypeVersion=2.
[INFO] Error stacktraces are turned on.
[INFO] Scanning for projects...
...
[INFO] BUILD SUCCESS ...
```

CLI archetype details

```
mvn --batch-mode ❶ -e archetype:generate ❷ \  
    \  
-DarchetypeGroupId=de.hdostuttgart.mi ❸ \  
-DarchetypeArtifactId=mi-naven-archetype-quickstart \  
-DarchetypeVersion=2.3 \  
    \  
-DgroupId=de.hdostuttgart.mi.sdl ❹ \  
-DartifactId=first \  
-Dversion=0.9
```

Generated project layout

```
> cd first          # Enter project directory
> find . -type f   # Search recursively for files
./pom.xml ❶
./src/main/java/de/hdm_stuttgart/mi/sd1/HighlightSample.java
./src/main/java/de/hdm_stuttgart/mi/sd1/Statistics.java
./src/main/java/de/hdm_stuttgart/mi/sd1/App.java ❷
./src/main/resources/log4j2.xml
./src/test/java/de/hdm_stuttgart/mi/sd1/AppTest.java
./README.md
```

Related exercises

Exercise 110: DNS inflicted groupid / package names clashes

Maven compile

```
> mvn compile
[INFO] Scanning for projects...
...
[INFO] Building first 0.9
...
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /na/goik/first/target/classes
[INFO] -----
[INFO] BUILD SUCCESS
```


Compilation file view

```
> find target/classes -type f  
./target/classes/de/hdm_stuttgart/mi/sd1/App.class  
...
```

Execution

```
> cd target/classes ❶  
> java de.hdm_stuttgart.mi.sd1.App ❷  
Hi there, let's have  
fun learning Java! ❸
```

- ❶ Change to base directory containing compiled Java™ classes.
- ❷ Application execution. Note:

Our App class is being prefixed by the package name de.hdm_stuttgart.mi.sd1 defined by the **groupId** parameter in Figure 4.101, “CLI archetype details”.

- ❸ The expected output result.

Note

Executing this particular class requires a configuration in our project's pom.xml file:

```
...  
<archive>  
  <manifest>  
    <mainClass>de.hdm_stuttgart.mi.sd1.test.ShowReachedPoints</mainClass>  
  </manifest>  
</archive> ...
```

Maven package

```
> mvn package
```

```
...
```

```
  T E S T S
```

```
...
```

```
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

```
...
```

```
[INFO] Building jar: /na/gok/first/target/first-0.9.jar
```

```
...
```

```
[INFO] Replacing /na/gok/first/target/first-0.9.jar with  
      /na/gok/first/target/first-0.9-shaded.jar
```

```
...
```

```
[INFO] BUILD SUCCESS
```

Executing Java™ archive first-0.9.jar

```
java -jar target/first-0.9.jar
```

```
Hi there, let's have
```

```
fun learning Java!
```

Remark: This will execute HelloWorld.class being contained in first-0.9.jar.

Related exercises

Exercise 111: Details on execution

Maven j avadoc: j avadoc

```
> mvn j avadoc: j avadoc
```

```
[INFO] Scanning for projects...
```

```
...
```

```
Generating /na/goik/First/target/site/api docs/all classes- no frame. html...
```

```
Generating /na/goik/First/target/site/api docs/index. html...
```

```
Generating /na/goik/First/target/site/api docs/overview summary. html...
```

```
Generating /na/goik/First/target/site/api docs/help- doc. html...
```

See e.g. class String documentation.

Maven clean

```
> mvn clean
```

```
...
```

```
[INFO]
```

```
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ first ---
```

```
[INFO] Deleting /na/goik/first/target
```

```
[INFO] -----
```

```
[
```

IntelliJ IDEA Maven support

- Built in Maven support in IntelliJ IDEA.
- MI supplementary archetypes require MI archetype configuration in `~/.m2/settings.xml`.

Adding MI Maven server

Adding MI Maven server

Adding MI Maven server

New MI archetype project

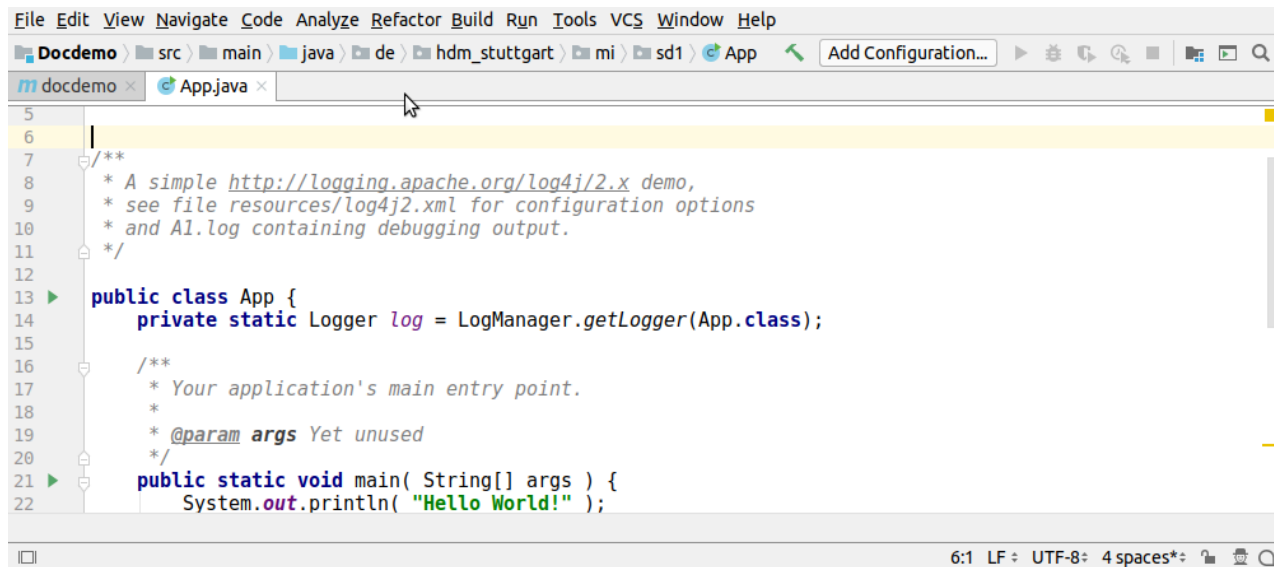
pom.xml content changes

Instance"
i-4.0.0.xsd">



Load Maven Changes Ctrl+Shift+O
Maven project structure has been changed. Load changes into IntelliJ IDEA to make it work correctly.

IntelliJ IDEA generating Javadoc™

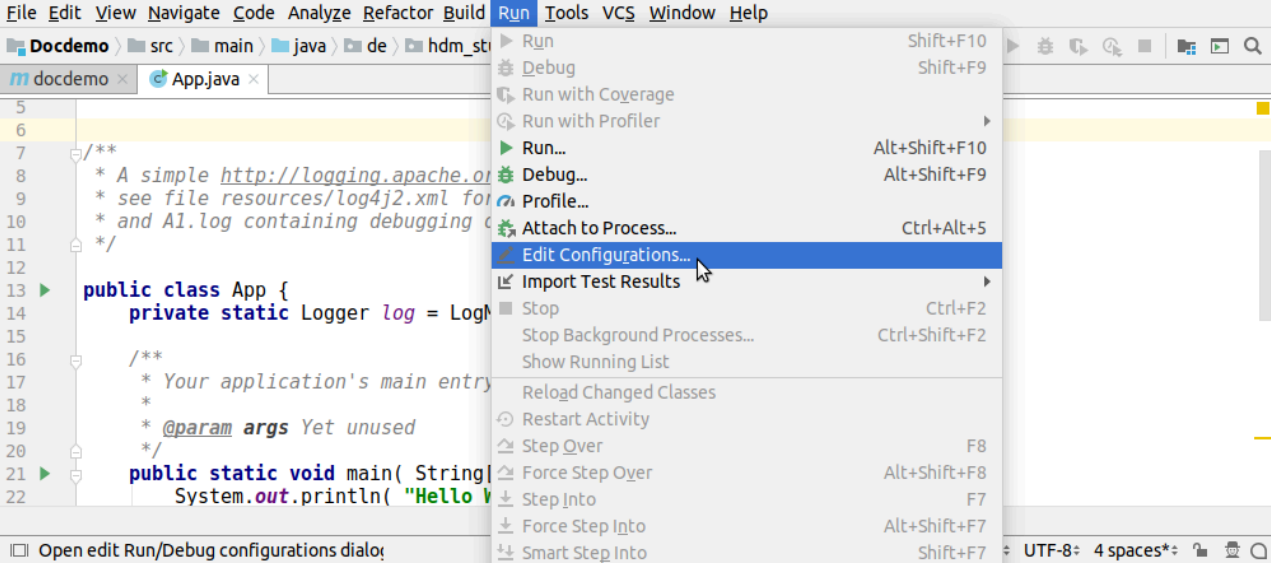


The screenshot shows the IntelliJ IDEA IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The breadcrumb navigation shows the path: Docdemo > src > main > java > de > hdm_stuttgart > mi > sd1 > App. The main editor window displays the source code for the class App.java. The code is as follows:

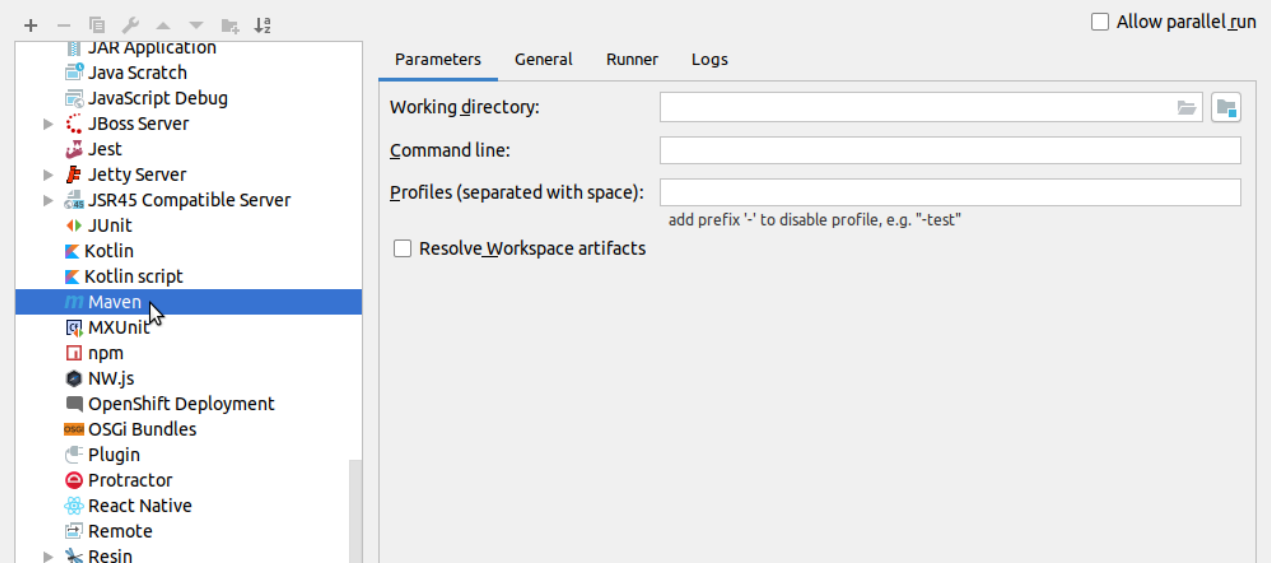
```
5  
6  
7  /**  
8   * A simple http://logging.apache.org/log4j/2.x demo,  
9   * see file resources/log4j2.xml for configuration options  
10  * and Al.log containing debugging output.  
11  */  
12  
13 public class App {  
14     private static Logger log = LogManager.getLogger(App.class);  
15  
16     /**  
17      * Your application's main entry point.  
18      *  
19      * @param args Yet unused  
20      */  
21     public static void main( String[] args ) {  
22         System.out.println( "Hello World!" );  
23     }  
24 }
```

The status bar at the bottom indicates the file encoding is UTF-8, line length is 6:1, and tab width is 4 spaces.

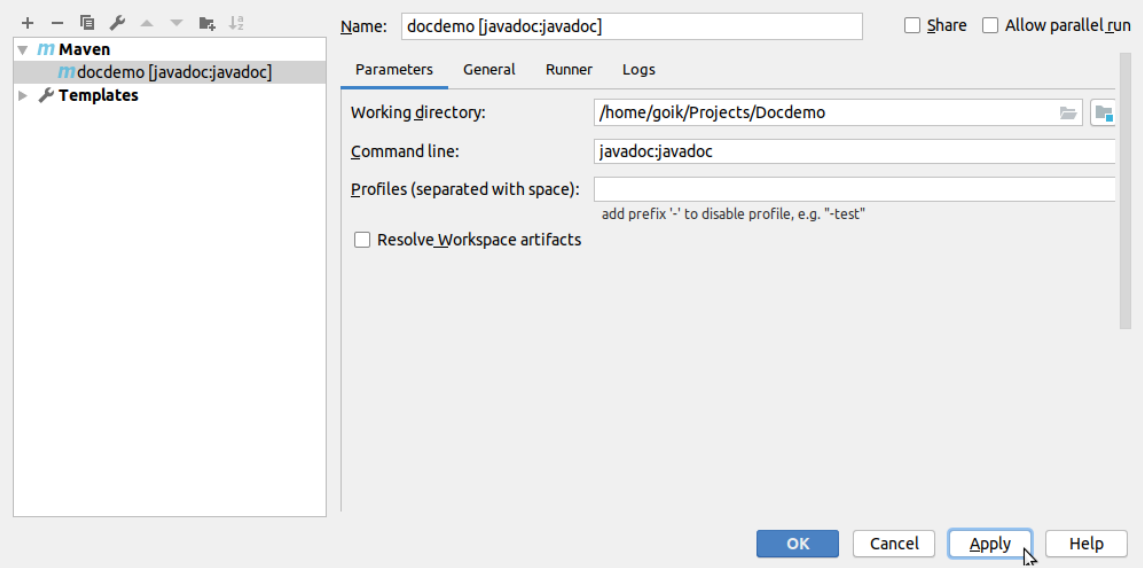
IntelliJ IDEA generating Javadoc™



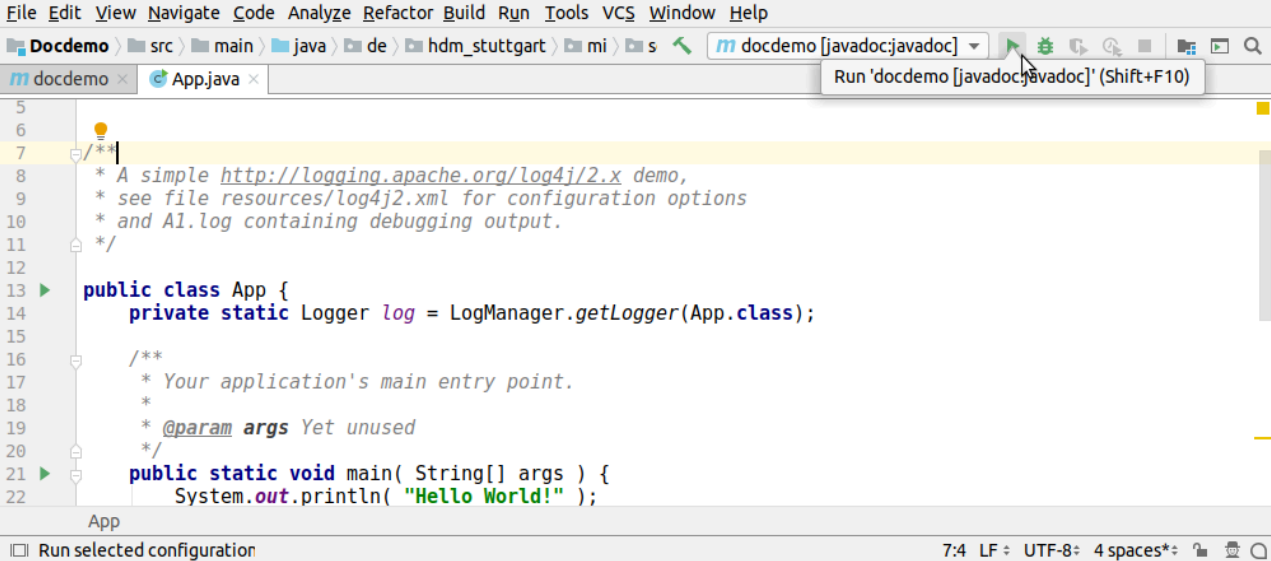
IntelliJ IDEA generating Javadoc™



IntelliJ IDEA generating Javadoc™



IntelliJ IDEA generating Javadoc™



The screenshot shows the IntelliJ IDEA interface with a Java file named 'App.java' open. The code is as follows:

```
5
6
7  /**
8   * A simple http://logging.apache.org/log4j/2.x demo,
9   * see file resources/log4j2.xml for configuration options
10  * and Al.log containing debugging output.
11  */
12
13 public class App {
14     private static Logger log = LogManager.getLogger(App.class);
15
16     /**
17     * Your application's main entry point.
18     *
19     * @param args Yet unused
20     */
21     public static void main( String[] args ) {
22         System.out.println( "Hello World!" );
23     }
24 }
```

The Javadoc comment on line 7 is highlighted in yellow. A tooltip above the Run button (a green play icon) displays the text: "Run 'docdemo [javadoc:javadoc]' (Shift+F10)". The status bar at the bottom shows "Run selected configuration", "7:4", "LF", "UTF-8", and "4 spaces*".

IntelliJ IDEA generating Javadoc™

The screenshot shows the IntelliJ IDEA interface during a Javadoc generation process. The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, and Help. The breadcrumb path is Docdemo > src > main > java > de > hdm_stuttgart > mi > s. The active file is docdemo [javadoc:javadoc].

The code editor shows the following Javadoc comment for the 'App' class:

```
5  
6  
7 /**  
8  * A simple http://logging.apache.org/log4j/2.x demo,  
9  * see file resources/log4j2.yml for configuration options  
App
```

The Run tool window shows the execution output:

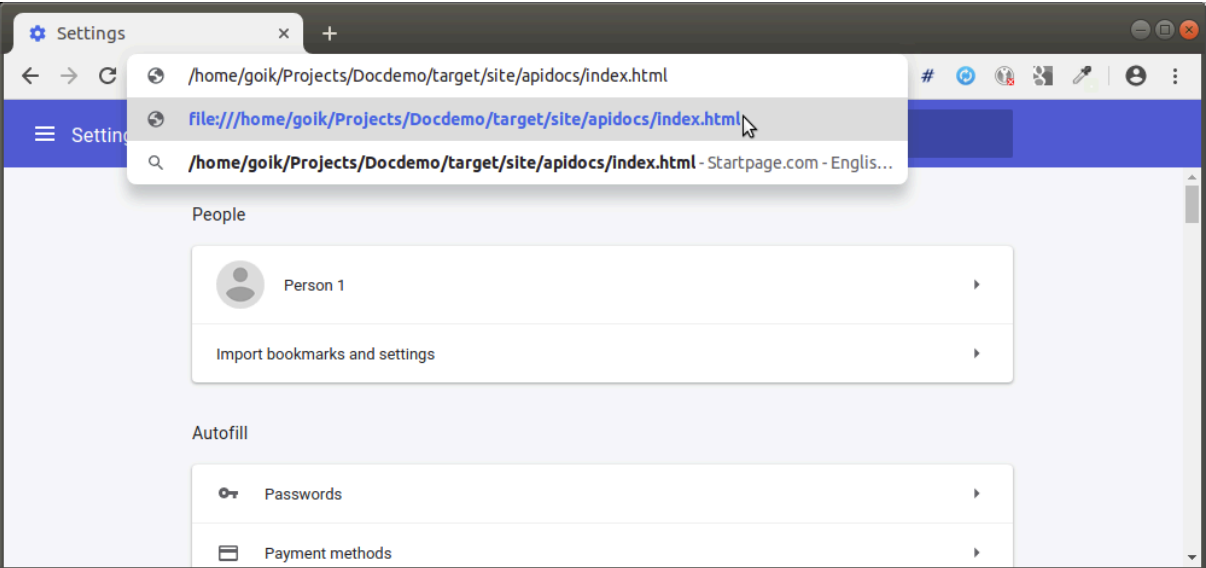
```
Run: m docdemo [javadoc:javadoc] x  
Generating /home/goik/Projects/Docdemo/target/site/apidocs/allclasses.html...  
Generating /home/goik/Projects/Docdemo/target/site/apidocs/allclasses.html...  
Generating /home/goik/Projects/Docdemo/target/site/apidocs/index.html...  
Generating /home/goik/Projects/Docdemo/target/site/apidocs/help-doc.html...  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.670 s  
[INFO] Finished at: 2019-06-16T21:34:42+02:00  
[INFO] Final Memory: 13M/74M  
[INFO] -----
```

A context menu is open over the output, with the following options:

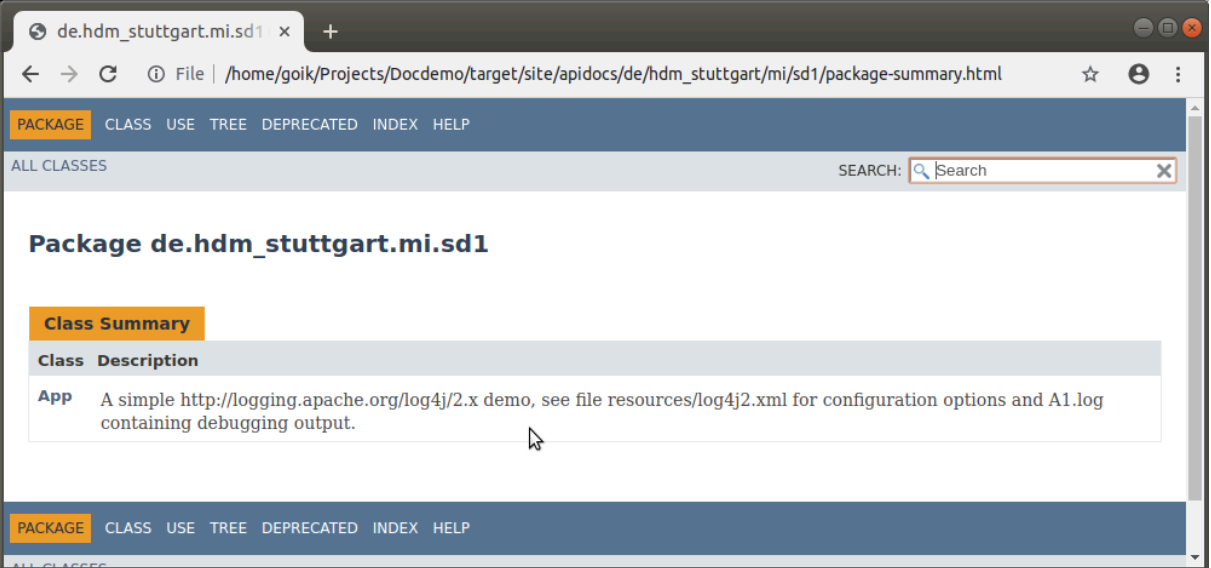
- Copy (Ctrl+C)
- Copy as Plain Text
- Compare with Clipboard
- Search with Google
- Fold Lines Like This
- Pause Output
- Create Gist...
- Clear All

The status bar at the bottom indicates: Copy selection to clipboard as plain te.. 7:4 LF UTF-8 4 spaces*+.

IntelliJ IDEA generating Javadoc™



IntelliJ IDEA generating Javadoc™



Related exercises

Exercise 112: Maximum and absolute value

Exercise 113: Factorial, the direct way

Exercise 114: Factorial, the recursive way

Exercise 115: Binomials, the recursive way

Exercise 116: The exponential

Exercise 117: Implementing .

Exercise 118: Summing up in a different order.

Recommended reading

- [Vogella2016]
- [TutorialsPointJunit]

Test categories

- **Unit test:** Test individual methods, classes and packages in isolation.
- **Integration Test:** Test a group of associated components/classes.
- **Acceptance / Functional Test:** Operate on a fully integrated system, testing against the user interface.
- **Regression Test:** Ensure system integrity after (implementation) change.
- **Load test:** Responsiveness vs. system load.

Example: Computing prime numbers

Informal problem specification:

A prime number is a whole number greater than 1 whose only factors are 1 and itself.

Sample values: 2, 3, 5, 7, 11, 13, 17, 23, ...

Unit test principle

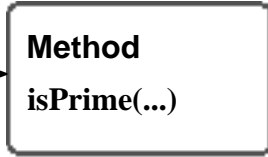
Testing method `isPrime(int n)`

Method
`isPrime(...)`

Unit test principle

Testing method `isPrime(int n)`

Input: 7



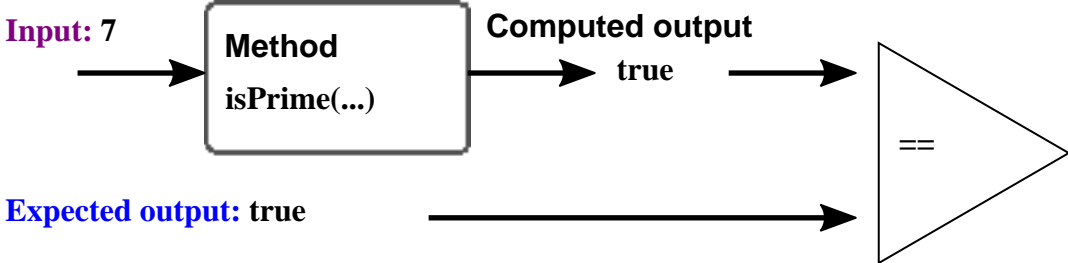
Unit test principle

Testing method `isPrime(int n)`



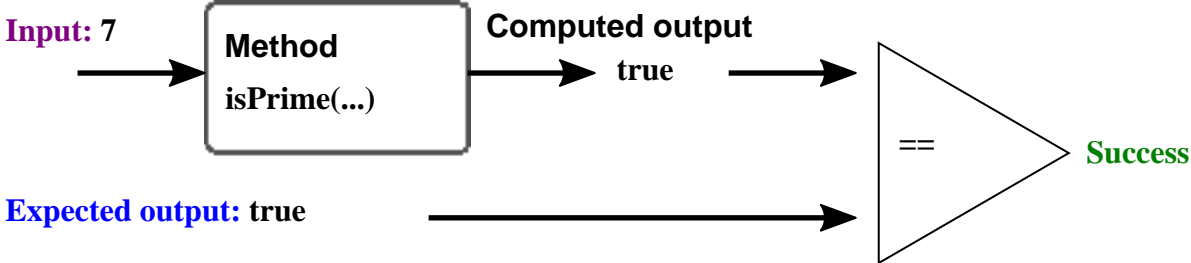
Unit test principle

Testing method isPrime(int n)



Unit test principle

Testing method `isPrime(int n)`



Unit test principle

Testing method `isPrime(int n)`

Input: 10



Method

`isPrime(...)`

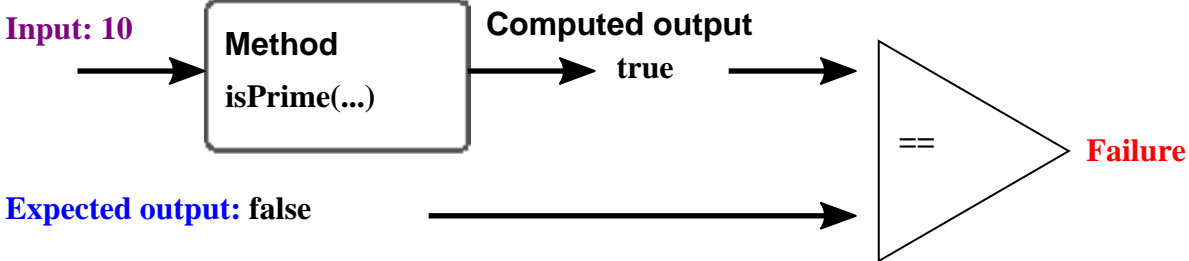
Unit test principle

Testing method `isPrime(int n)`



Unit test principle

Testing method isPrime(int n)



Test driven development

First write tests, then implement.

Steps in Unit Testing

1. Specify but not yet implement classes / methods.
2. Write skeleton (dummy) implementations.
3. Write corresponding unit tests.
4. Implement skeleton.
5. Test your implementation.

Step 1 + 2: Specify method, write skeleton

```
/**
 * Dealing with prime numbers.
 */
public class Prime {
    /**
     * Check whether a given value is prime or not ❶
     * @param value A positive value
     * @return true if and only if value is a prime number.
     */
    public static boolean isPrime(int value) {
        return true ❷; //TODO Dummy value to be implemented correctly
    }
}
```

Execution yet being flawed

```
for (int i = 1; i < 20; i++) {  
    System.out.println(i + " is " + (Prime.isPrime(i) ? " a " : " not a ")  
        + " prime number");  
}
```

```
1 is a prime number  
2 is a prime number  
3 is a prime number  
4 is a prime number  
5 is a prime number  
...
```

Sample test data

Input	Expected output	Input	Expected output
1	false	7	true
2	true	8	false
3	true	9	false
4	false	10	false
5	true	11	true
6	false	12	false

Step 3: Junit based specification test

```
public class PrimeTest {  
  
    @Test ❶ public void test_1_isNotPrime() {  
        Assert.assertFalse(Prime.isPrime(1));  
    }  
  
    @Test ❶ public void test_2_isPrime() {  
        Assert.assertTrue(Prime.isPrime(2));  
    }  
  
    void someOrdinaryMethod() ❷ {...}  
...  
}
```

Junit skeleton test result (Maven CLI)

```
goik@goiki Prime_v01> mvn test
```

```
...
```

```
Running de.hdm.stuttgart.mi.sd1.PrimeTest
```

```
Tests run: 2, Failures: 1, Errors: 0, Skipped: 0,
```

```
Time elapsed: 0.065 sec <<< FAILURE!
```

```
...
```

```
test_1_isNotPrime(de.hdm.stuttgart.mi.sd1.PrimeTest)
```

```
Time elapsed: 0.001 sec <<< FAILURE!
```

```
java.lang.AssertionError
```

```
at org.junit.Assert.fail(Assert.java:86)
```

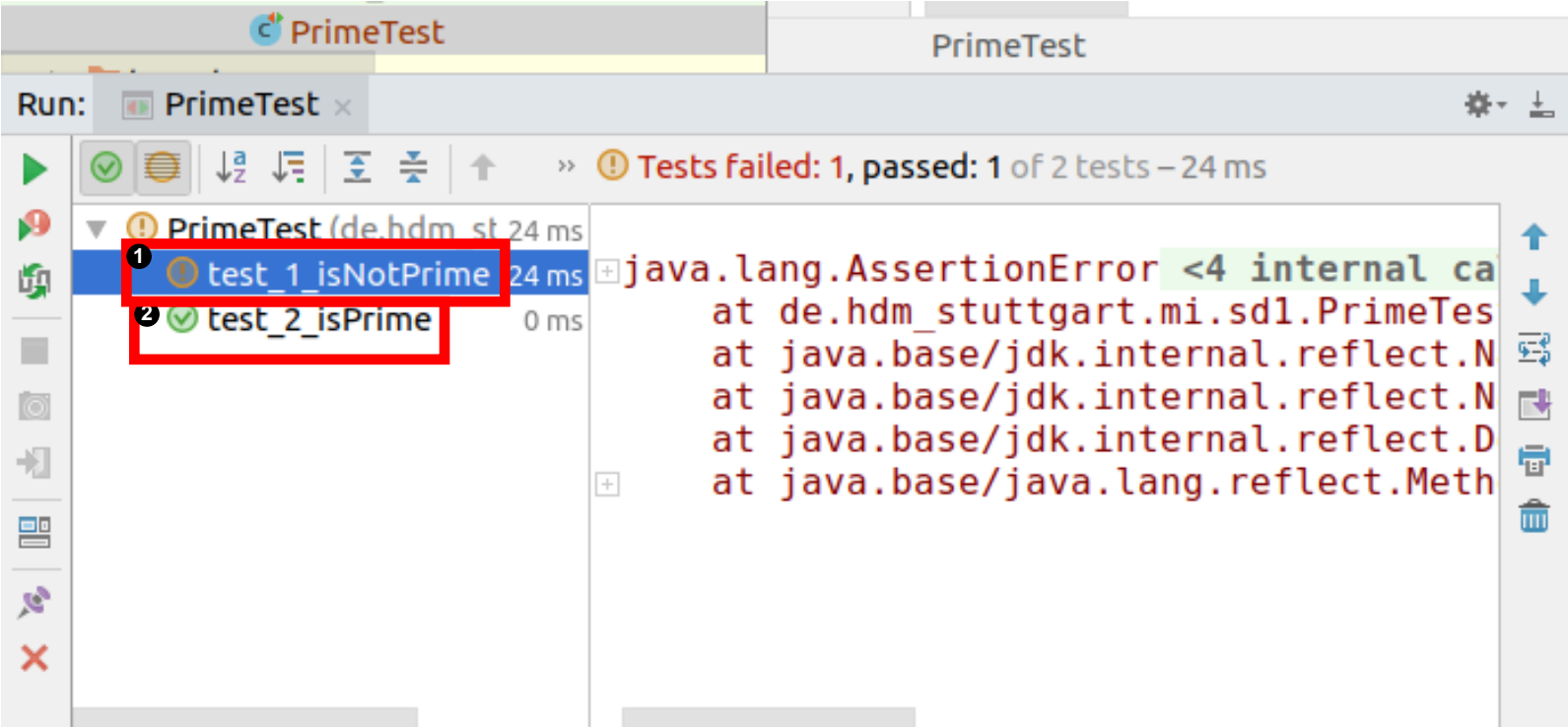
```
at org.junit.Assert.assertTrue(Assert.java:41)
```

```
at org.junit.Assert.assertFalse(Assert.java:64)
```

```
at org.junit.Assert.assertFalse(Assert.java:74)
```

```
...
```


JUnit skeleton test result (IDE)



Step 3: Providing more prime tests

```
@Test public void testPrimes() {
    Assert.assertTrue(Prime.isPrime(3));
    Assert.assertTrue(Prime.isPrime(5));
    Assert.assertTrue(Prime.isPrime(7));
    Assert.assertTrue(Prime.isPrime(11));
    ... }

@Test public void testOddNonPrimes() {
    Assert.assertFalse(Prime.isPrime(9));
    Assert.assertFalse(Prime.isPrime(15));
    Assert.assertFalse(Prime.isPrime(21)); ... }
```

Step 3: Prime mass testing

```
@Test public void testEvenNonPrimes() {  
    for (int i = 2; i < 100; i++) {  
        Assert.assertFalse(Prime.isPrime(2 * i));  
    }  
}
```

Step 4: Implement skeleton

```
public static boolean isPrime(int value) {  
    for (int i = 2; i < value; i++) {  
        if (0 == value % i) { // i divides value  
            return false;  
        }  
    }  
    return value != 1;  
}
```

Step 5: Testing our first implementation

```
goik@goiki Prime_v01> mvn test
```

```
...
```

```
-----  
T E S T S  
-----
```

```
Running de.hdunstuttgart.misdl.PrimeTest
```

```
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.055 sec
```

```
Results :
```

```
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

Implementation observation

101 / 2 = 50 remainder 1
101 / 3 = 33 remainder 2
101 / 4 = 25 remainder 1
101 / 5 = 20 remainder 1
101 / 6 = 16 remainder 5
101 / 7 = 14 remainder 3
101 / 8 = 12 remainder 5
101 / 9 = 11 remainder 2
101 / 10 = 10 remainder 1
101 / 11 = 9 remainder 2
101 / 12 = 8 remainder 5
101 / 13 = 7 remainder 10

...

Changing the implementation

Big performance gain:

```
public static boolean isPrime(int value) {  
    for (int i = 2; i * i < value; i++) {  
        if (0 == value % i) {  
            return false;  
        }  
    }  
    return value != 1;  
}
```

Regression test

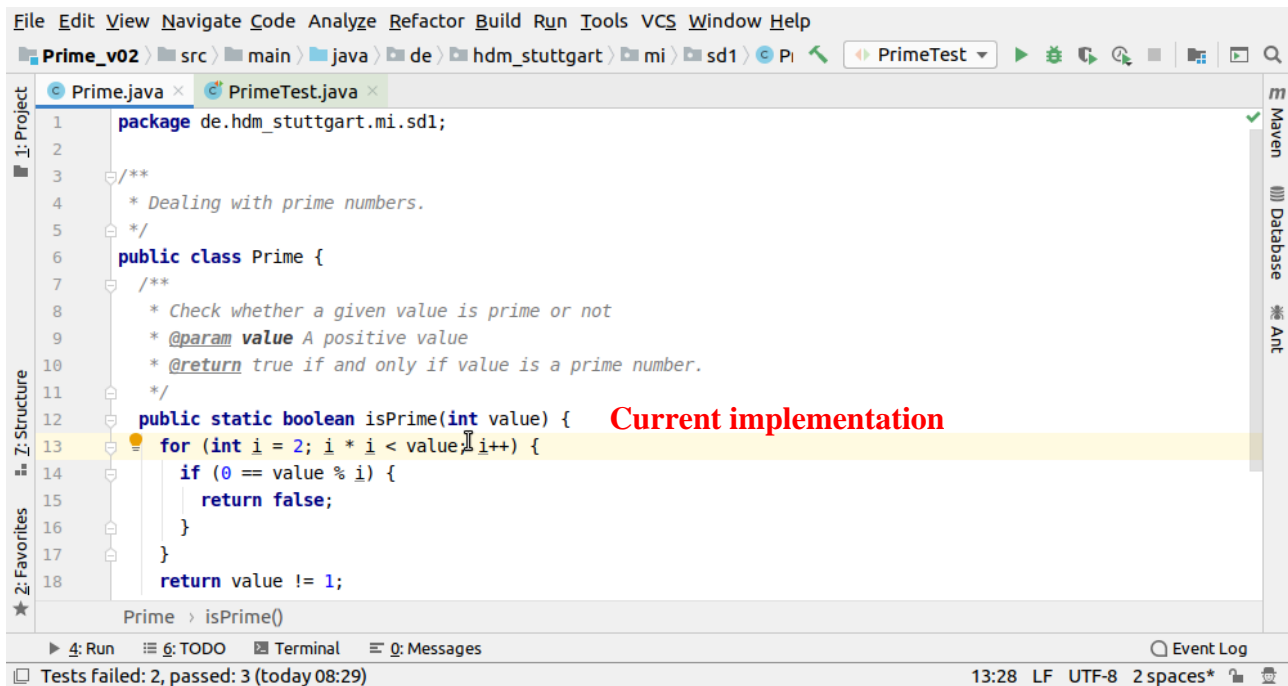
The screenshot shows the Run window of an IDE. The package path is `de.hdm_stuttgart.mi.sd1`. The class being run is `PrimeTest`. The test results are as follows:

Test Name	Duration	Status
<code>testOddNonPrimes</code>	11 ms	Failed
<code>test_1_isNotPrime</code>	0 ms	Passed
<code>test_Primes</code>	0 ms	Passed
<code>test_2_isPrime</code>	0 ms	Passed
<code>testEvenNonPrimes</code>	2 ms	Failed

The error details for the failed tests are:

```
java.lang.AssertionError: Testing 4
<3 internal calls>
at de.hdm_stuttgart.mi.sd1.PrimeTest.testEvenNonPrimes(PrimeTest.java:65)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(NativeMethodAccessorImpl.java:62)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:69)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:566)
```


Systematic error debugging

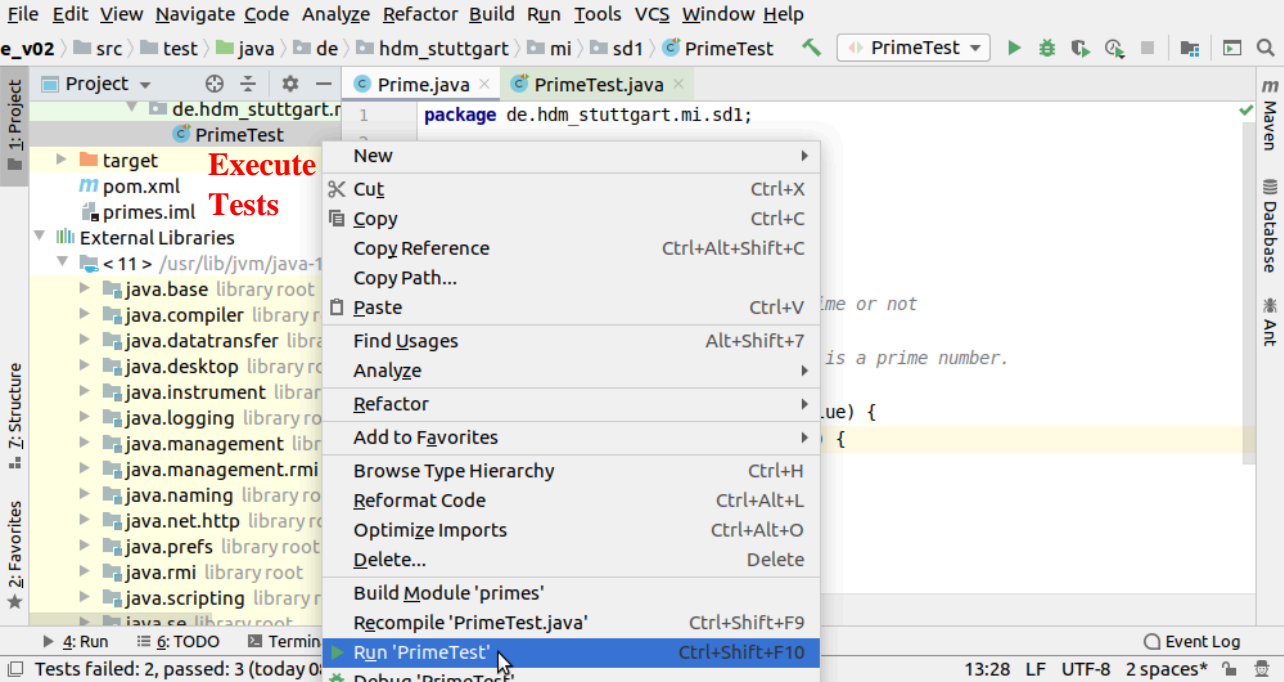


The screenshot shows an IDE window with the following content:

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Prime_v02 src main java de hdm_stuttgart mi sd1 PrimeTest PrimeTest
Prime.java PrimeTest.java
1 package de.hdm_stuttgart.mi.sd1;
2
3 /**
4  * Dealing with prime numbers.
5  */
6 public class Prime {
7  /**
8  * Check whether a given value is prime or not
9  * @param value A positive value
10 * @return true if and only if value is a prime number.
11 */
12 public static boolean isPrime(int value) { Current implementation
13     for (int i = 2; i * i < value; i++) {
14         if (0 == value % i) {
15             return false;
16         }
17     }
18     return value != 1;
19 }
Prime > isPrime()
```

At the bottom of the IDE, the status bar displays: 4: Run, 6: TODO, Terminal, 0: Messages, Event Log, Tests failed: 2, passed: 3 (today 08:29), 13:28 LF UTF-8 2 spaces*

Systematic error debugging



Systematic error debugging

The screenshot shows an IDE interface with the following components:

- Code Editor:** Displays the `isPrime` method in `PrimeTest.java`. The loop `for (int i = 2; i * i < value; i++)` is highlighted in yellow. A red text annotation **Two failing tests** is placed over the code.
- Run Window:** Shows the execution of `PrimeTest`. The summary indicates **Tests failed: 2, passed: 3 of 5 tests - 6 ms**. The failed tests are `testOddNonPrimes` (5 ms) and `testEvenNonPrimes` (1 ms).
- Stack Trace:** The error is a `java.lang.AssertionError` with the message `<4 internal calls>`. The stack trace includes:
 - `at de.hdm_stuttgart.mi.sd1.PrimeTest.testOddNonPrimes(PrimeTest.java:63) <19 internal calls>`
 - `at com.intellij.rt.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:230)`
 - `at com.intellij.rt.junit.JUnit4TestRunner.prepareStreamsAndStart(JUnit4TestRunner.java:230)`
 - `at com.intellij.rt.junit.JUnit4TestRunner.main(JUnit4TestRunner.java:58)`

Systematic error debugging

The screenshot shows an IDE window with the following components:

- Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Explorer:** Shows a project structure with folders like 'target', 'pom.xml', 'primes.iml', and 'External Libraries'.
- Code Editor:** Displays the source code for `PrimeTest.java`. A red dot indicates a breakpoint is set on line 63. A red annotation "Set breakpoint on first failing test" points to this line. The code includes several `Assert.assertFalse` calls for prime numbers 9, 15, 21, 25, and 27.
- Run Console:** Shows the execution of the `PrimeTest` class. It reports "Tests failed: 2, passed: 3 of 5 tests - 6 ms". The failed tests are `testOddNonPrimes` (5 ms) and `testEvenNon` (1 ms). The stack trace for the failure shows an `AssertionError` at line 63 of `PrimeTest.java`.
- Status Bar:** Shows "Tests failed: 2, passed: 3 (a minute ago)" and "63:12 LF UTF-8 2 spaces*".

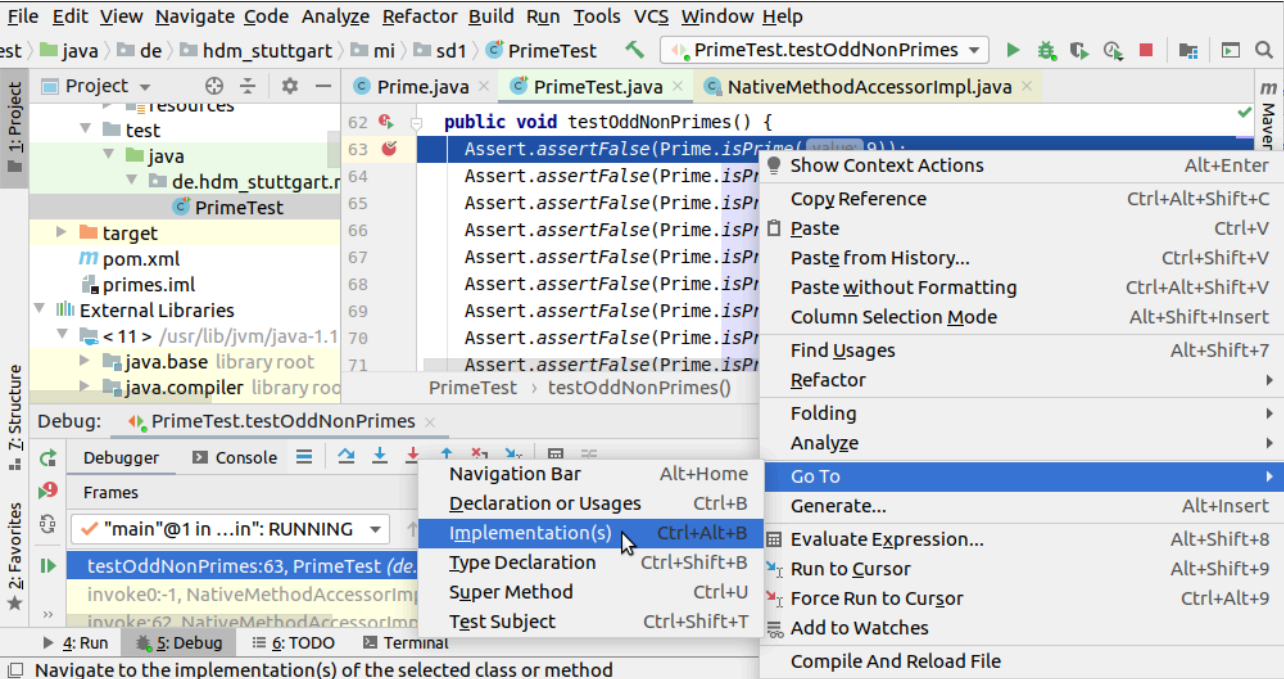
Systematic error debugging

The screenshot shows an IDE window with the following components:

- Project Explorer:** Shows the project structure with 'test' and 'java' folders. The file 'PrimeTest.java' is selected.
- Code Editor:** Displays the code for 'testOddNonPrimes()' with several 'Assert.assertFalse' calls. Line 63 is highlighted in red, indicating a failure.
- Run Console:** Shows the test execution results: 'Tests failed: 1 of 1 test - 7 ms'. The failed test 'testOddNonPrimes' is expanded, showing the error message: 'java.lang.AssertionError <4 internal calls>'. A context menu is open over the error, with 'Debug 'testOddNonPrimes()'' selected.
- Terminal:** Shows the output of the test run: 'Tests failed: 1, passed: 0 (moments)'. The status bar at the bottom indicates '63:12 LF UTF-8 2 spaces*'.

Run first test only in debug mode

Systematic error debugging



Systematic error debugging

The screenshot shows an IDE with the following components:

- Code Editor:** Displays the `Prime` class with a breakpoint at line 13: `for (int i = 2; i * i < value; i++) {`. The code includes a comment: `* @return true if and only if value is a prime number.`
- Debugger Window:** Shows the current frame as `"main"@1 in ...in": RUNNING`. The variable `this` is shown as `{PrimeTest@1139}`.
- Buttons:** A tooltip for the `Resume Program` button (F9) is visible.

Annotations: Red text on the right side of the code editor reads: **Create second breakpoint at method start. Then hit »Resume Program«**

Systematic error debugging

The screenshot shows an IDE window with the following components:

- Project Explorer:** Shows the project structure with folders like 'resources', 'test', 'java', and 'target'. The file 'PrimeTest.java' is selected.
- Code Editor:** Displays the source code for the 'Prime' class. The method 'isPrime' is highlighted, and the current execution point is at line 13: `for (int i = 2; i * i < value; i++) {`. The variable 'value' is shown as 9.
- Debugger:** The 'Step Over' button (F8) is highlighted. The 'Frames' window shows the current stack frame: `isPrime:13, Prime (de.hdm_stuttgart.mi.sd1)`. The 'Variables' window shows `value = 9`.
- Status Bar:** Shows the current time as 13:11, encoding as UTF-8, and 2 spaces.

Start stepping through your code watching variables changing.

Systematic error debugging

The screenshot shows an IDE window with the following components:

- Project Explorer:** Shows the project structure with folders like 'resources', 'test', 'java', and 'target'.
- Code Editor:** Displays the `PrimeTest.java` file. The code is as follows:

```
10  * @return true if and only if value is a prime number.
11  */
12  public static boolean isPrime(int value) { value: 9
13  for (int i = 2; i * i < value; i++) { i: 2 value: 9
14  if (0 == value % i) {
15  return false;
16  }
17  }
18  return value != 1;
19  }
```
- Debugger:** Shows the current execution state. The 'Frames' pane lists:
 - "main"@1 in ...in": RUNNING
 - isPrime:13, Prime (de.hdm_stuttgart.mi.sd1)
 - testOddNonPrimes:63, PrimeTest (de.hdm_stu...
 - invoke0:-1, NativeMethodAccessorImpl (jdk.int...
- Variables:** Shows the current values of variables:
 - value = 9
 - i = 2
- Annotations:** A red text annotation on the right side of the code editor reads: **Current value: i == 2**

Systematic error debugging

The screenshot shows an IDE window with the following components:

- Project Explorer:** Shows the project structure with folders like 'resources', 'test', 'java', and 'target'.
- Code Editor:** Displays the code for the `isPrime` method in `PrimeTest.java`. The code is as follows:

```
12 public static boolean isPrime(int value) { value: 9
13     for (int i = 2; i * i < value; i++) {
14         if (0 == value % i) {
15             return false;
16         }
17     }
18     return value != 1; value: 9
19 }
20 }
```
- Debugger:** Shows the current state of the program. The `value` variable is set to 9. The current frame is `isPrime:18, Prime (de.hdm_stuttgart.mi.sd1)`.
- Annotations:** A red arrow points to the `for` loop, and a red text overlay says "Whoops: We did not check for i == 3". Another red text overlay says "=> premature loop termination!".

Systematic error debugging

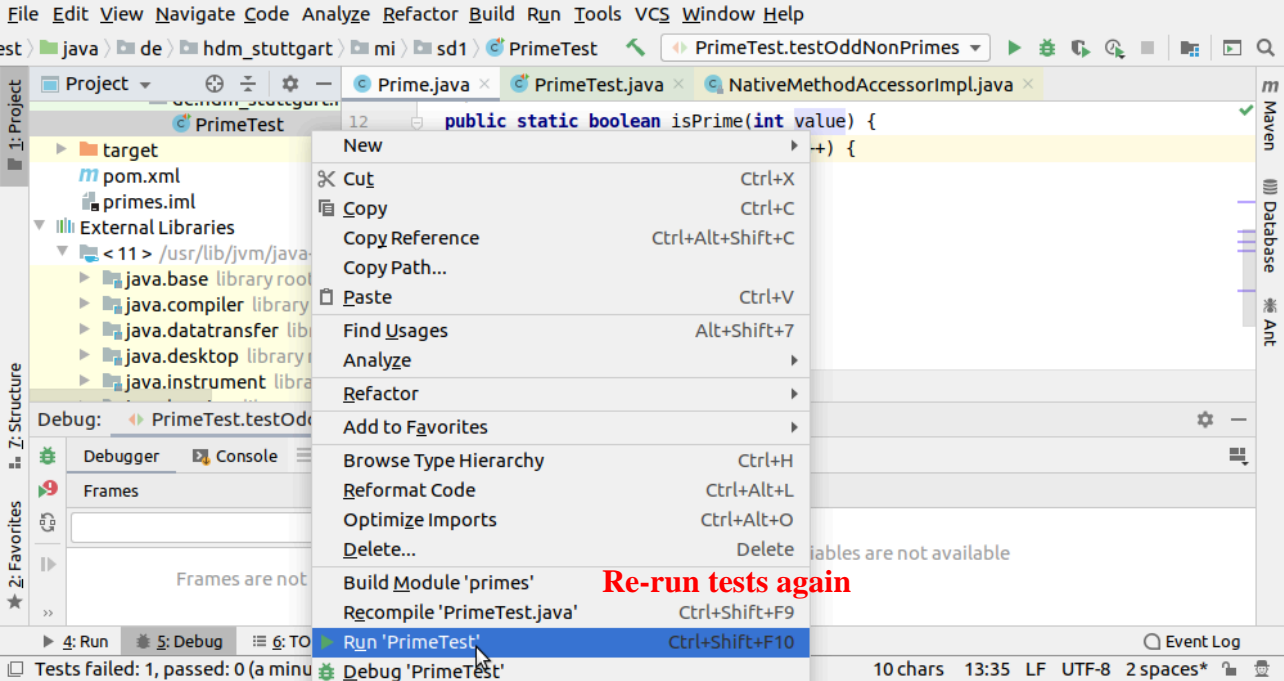
The screenshot shows an IDE window with the following components:

- Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Explorer:** Shows a project structure with folders like 'resources', 'test', 'java', and 'de.hdm_stuttgart.r'. A file 'PrimeTest' is highlighted.
- Code Editor:** Displays the following Java code:

```
12 public static boolean isPrime(int value) {  
13     for (int i = 2; i * i <= value; i++) {  
14         if (0 == value % i) {  
15             return false;  
16         }  
17     }  
18     return value != 1;  
19 }  
20 }
```

The loop condition `i * i <= value` is highlighted in yellow. A red annotation **Correcting loop terminating expression** is placed over the `<=` operator.
- Debugger:** Shows the current execution context: `PrimeTest.testOddNonPrimes`. The `Frames` and `Variables` panels are empty, with the message "Variables are not available" and "Frames are not available".
- Status Bar:** Shows "Tests failed: 1, passed: 0 (a minute ago)", "10 chars", "13:35", "LF", "UTF-8", "2 spaces*", and "Event Log".

Systematic error debugging



Systematic error debugging

The screenshot shows an IDE with the following components:

- Code Editor:** Displays the `isPrime` method in `PrimeTest.java`. The code is as follows:

```
8  * Check whether a given value is prime or not
9  * @param value A positive value
10 * @return true if and only if value is a prime number.
11 */
12 public static boolean isPrime(int value) {
13     for (int i = 2; i * i <= value; i++) {
14         if (0 == value % i) {
15             return false;
16         }
17     }
18     return true;
19 }
```
- Test Runner:** Shows a summary of test results for `PrimeTest`. All tests passed:
 - PrimeTest (de.hdm_stuttga) 1 ms
 - testOddNonPrimes 1 ms
 - test_1_isNotPrime 0 ms
 - test_Primes 0 ms
 - test_2_isPrime 0 ms
 - testEvenNonPrimes 0 msThe message "Process finished with exit code 0" is displayed.
- Bottom Status Bar:** Shows "Tests passed: 5 (moments ago)", "10 chars", "13:35", "LF", "UTF-8", and "2 spaces*".

Success: Second test failure vanished as well.

Error correction in detail

```
public static boolean isPrime(int value) {  
  
    //for (int i = 2; i * i < value; i++) {  
    for (int i = 2; i * i <= value; i++) {  
        if (0 == value % i) {  
            return false;  
        }  
    }  
    return value != 1;  
}
```

Available comparison methods

- `assertEquals([String message], ...)`
- `assertArrayEquals([String message], ...)`
- `assertFalse([String message], ...)`
- `assertNotEquals([String message], ...)`
- `assertNull([String message], ...)`
- `fail([String message], ...)`

Caution comparing float / double !!

```
6  /**
7   * Comparing double values
8   */
9  public class doubleCompareTest {
10     /**
11     * Comparing 3.6 and 3.6 ?
12     */
13     @Test
14     public void test1_3() {
15         Assert.assertEquals( expected: 3.6, actual: 3 * 1.2 );
16     }
17 }
18
```

'assertEquals(double, double)' is deprecated [more...](#) (Ctrl+F1)

Weird arithmetics?

```
java.lang.AssertionError: Use assertEquals(expected, actual, delta)
    to compare floating-point numbers
```

```
at org.junit.Assert.assertEquals(Assert.java:656)
```

```
at qq.doubleCompareTest.test1_3(doubleCompareTest.java:15)
```

```
...
```

Limited representation precision

```
System.out.println("3.6 - 3 * 1.2 == " + (3.6 - 3 * 1.2));
```

Result:

```
3.6 - 3 * 1.2 == 4.440892098500626E-16
```

Solving the issue

```
public class doubleCompareTest {
    static final double delta = 1E-15;
    /**
     * Comparing 3.6 and 3 * 1.2 within delta's limit
     */
    @Test
    public void test1_3() {
        Assert.assertEquals(3.6, 3 * 1.2, delta);
    }
}
```

Related exercises

Exercise 119: Summing up integers to a given limit

Exercise 120: Summing up, the better way

The @Test annotation

```
public
@interface Test {
...
}
```

- @interface defining an annotation.
- Purpose: Adding meta information for automated detection of test methods.

The Assert class

```
public class Assert {  
  
    public static void assertTrue(  
        String message, boolean condition) { ... }  
  
    public static void assertEquals(  
        long expected, long actual) { ... }  
    ...  
}
```

Importing dependencies

```
<project ... >
...
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13</version>

      <scope>test</scope>
    </dependency>
    ...
  </dependencies>
  ...
</project >
```

Local: /home/goik/.m2/repository/junit/junit/4.13/junit-4.13.jar

Remote: <https://mvnrepository.com/artifact/junit/junit/4.13>

Dependency archive content

```
> jar -tf junit-4.13.jar
```

```
META-INF/
```

```
META-INF/MANIFEST.MF
```

```
org/
```

```
org/junit/
```

```
...
```

```
org/junit/Assert.class
```

```
...
```


Related exercises

Exercise 121: Turning seconds into weeks, part 2

Exercise 122: Example: A class representing fractions

Value vs. reference type variables

Value type: Holding data in associated memory location

- byte
- short

- int
- long

- float
- double

- char
- boolean

Reference type:
Holding a reference to an object

Array or class instances i.e. String, java.util.Scanner etc.

Different behaviour!

Value type	<pre>int a = 1; int b = a; a = 5; System.out.println("a=" + a); System.out.println("b=" + b);</pre>	<pre>a=5 b=1</pre>
Reference type	<pre>StringBuffer r = new StringBuffer("Joe"); StringBuffer s = r; r.append(" Simpson"); System.out.println("r=" + r); System.out.println("s=" + s);</pre>	<pre>r=Joe Simpson s=Joe Simpson</pre>

Value variable Details

Stack

Heap

```
int a = 1;
```

No objects
involved

```
int b = a;
```

```
a = 5;
```

```
System.out.println("a="+a);
```

```
System.out.println("b="+b);
```

Create int variable a on stack.

Value variable Details

	Stack	Heap
<code>int a = 1;</code>	<code>a: 1</code>	No objects involved
<code>int b = a;</code>		
<code>a = 5;</code>		
<code>System.out.println("a="+a);</code> <code>System.out.println("b="+b);</code>		

Create second int variable b on stack, initialize with a's value.

Value variable Details

```
int a = 1;
```

```
int b = a;
```

```
a = 5;
```

```
System.out.println("a="+a);
```

```
System.out.println("b="+b);
```

Stack



Heap

No objects
involved

Assign new value 5 to variable a.

Independent variable b does not change.

Value variable Details

	Stack	Heap
<code>int a = 1;</code>	<code>a: 5</code>	No objects involved
<code>int b = a;</code>	<code>b: 1</code>	
<code>a = 5;</code>		
<code>System.out.println("a="+a);</code> <code>System.out.println("b="+b);</code>	Print result showing: <code>a=5</code> <code>b=1</code>	

Reference variable Details

Stack

Heap

```
StringBuffer r =  
    new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);
```

```
System.out.println("s="+s);
```

Create StringBuffer instance on heap.

Reference variable Details

Stack

Heap

```
StringBuffer r =  
    new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);
```

```
System.out.println("s="+s);
```

Joe



Assign heap reference to stack variable r.

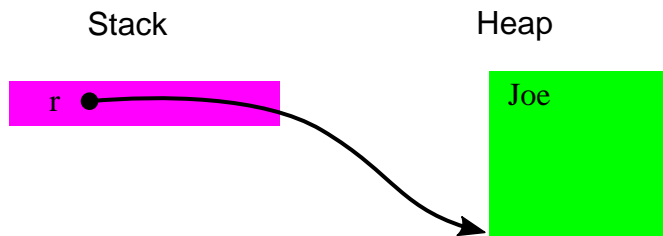
Reference variable Details

```
StringBuffer r =  
    new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);  
System.out.println("s="+s);
```



Create second variable holding a copy of variable r's heap reference: Both variables r and s point to the common StringBuffer instance.

Reference variable Details

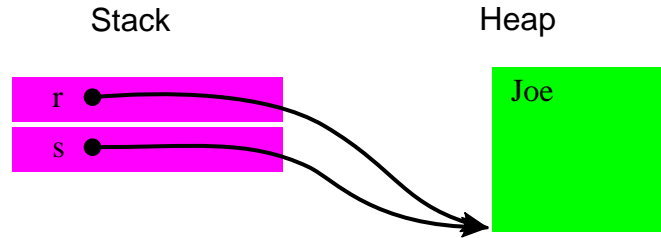
```
StringBuffer r =  
    new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);
```

```
System.out.println("s="+s);
```



Use variable r's reference for appending
" Simpson" to existing StringBuffer
instance.

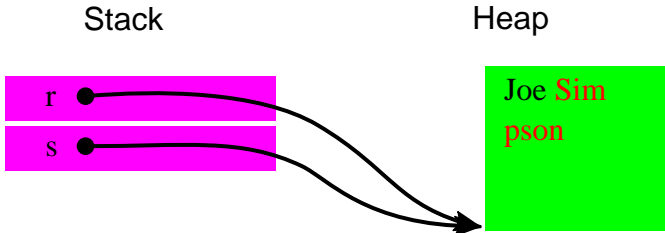
Reference variable Details

```
StringBuffer r =  
  new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);  
System.out.println("s="+s);
```



Print common StringBuffer instance two times using both variable references a and b in sequence. Result:

Joe Simpson
Joe Simpson

Only «call-by-value» in Java™

```
public static void main(String[] args) {
    int value = 3;
    System.out.println(
        "Before print DuplicateValue: "
        + value);
    printDuplicateValue(value);
    System.out.println(
        "After print DuplicateValue: "
        + value);
}
static void printDuplicateValue(int n) {
    n = 2 * n;
    System.out.println(
        "print DuplicateValue: " + n);
}
```

```
Before print DuplicateValue: 3
print DuplicateValue: 6
After print DuplicateValue: 3
```

«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
    printDuplicateValue: "  
    + value);  
  printDuplicateValue(value);  
  ...println("After  
    printDuplicateValue: "  
    + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
    "printDuplicateValue: " + n);  
}
```

Stack

main(...)

Create int variable assigning initial value 3.

«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
  printDuplicateValue: "  
  + value);  
  printDuplicateValue(value);  
  ...println("After  
  printDuplicateValue: "  
  + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
  "printDuplicateValue: " + n);  
}
```

main(...)

Stack

value: 3

Print content of variable value.
Result: 3

«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
    printDuplicateValue: "  
    + value);  
  printDuplicateValue(value);  
  ...println("After  
    printDuplicateValue: "  
    + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
    "printDuplicateValue: " + n);  
}
```

main(...)

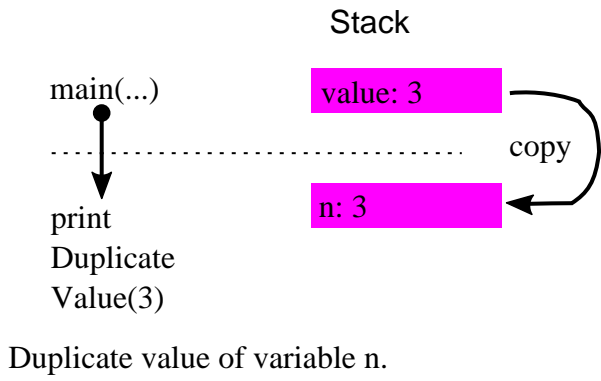
Stack

value: 3

Call printDuplicateValue(3):
Opening stack frame. Copy
variable value's content to
method's parameter variable n

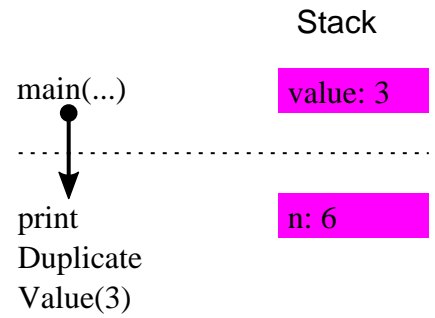
«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
  printDuplicateValue: "  
    + value);  
  printDuplicateValue(value);  
  ...println("After  
  printDuplicateValue: "  
    + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
  "printDuplicateValue: " + n);  
}
```



«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
    printDuplicateValue: "  
    + value);  
  printDuplicateValue(value);  
  ...println("After  
    printDuplicateValue: "  
    + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
    "printDuplicateValue: " + n);  
}
```



Print value of variable n.
Result: 6
Return to `main(...)` method releasing stack frame.

«call-by-value» details

```
...main(String[] args) {  
  int value = 3;  
  ...println("Before  
    printDuplicateValue: "  
    + value);  
  printDuplicateValue(value);  
  ...println("After  
    printDuplicateValue: "  
    + value);  
} ... printDuplicateValue(int n){  
  n = 2 * n;  
  ...println(  
    "printDuplicateValue: " + n);  
}
```

main(...)

Stack

value: 3

Print variable value's content.
Result: 3

«call-by-reference» for objects?

```
public static void main(String[] args) {
    StringBuffer buffer = new StringBuffer("My");
    System.out.println("Before duplicateString: "
        + buffer);
    duplicateString(buffer);
    System.out.println("After duplicateString: "
        + buffer);
}
static void duplicateString(StringBuffer b) {
    b.append(b); // Append self
}
```

Before duplicateString: My
After duplicateString: MyMy

«call-by-reference» details

	Stack	Heap
<pre>... main(...) { StringBuffer buffer= new StringBuffer("My"); ...println("Before duplicateString:" + buffer); duplicateString(buffer); ...println("After duplicateString:" + buffer); } ...duplicateString(StringBuffer b){ b.append(b);}</pre>	<pre>main(...)</pre>	

Create StringBuffer instance on heap.

«call-by-reference» details

```
... main(...) {  
  StringBuffer buffer=  
  new StringBuffer("My");  
  ...println(  
  "Before duplicateString:"  
    + buffer);  
  duplicateString(buffer);  
  ...println(  
  "After duplicateString:"  
    + buffer);  
}  
...duplicateString(  
  StringBuffer b){  
  b.append(b);}
```

main(...)

Stack

Heap

"My"

Create reference variable buffer on stack initializing with StringBuffer heap reference.

«call-by-reference» details

```
... main(...) {  
  StringBuffer buffer=  
    new StringBuffer("My");  
  ...println(  
    "Before duplicateString:"  
    + buffer);  
  duplicateString(buffer);  
  ...println(  
    "After duplicateString:"  
    + buffer);  
}  
...duplicateString(  
  StringBuffer b){  
  b.append(b);}
```

main(...)

Stack

buffer

Heap

"My"

Using variable buffer's reference
to print heap instance.
Result: "My"

«call-by-reference» details

```
... main(...) {  
  StringBuffer buffer=  
    new StringBuffer("My");  
  ...println(  
    "Before duplicateString:"  
    + buffer);  
  duplicateString(buffer);  
  ...println(  
    "After duplicateString:"  
    + buffer);  
}  
...duplicateString(  
  StringBuffer b){  
  b.append(b);}
```

main(...)

Stack

buffer

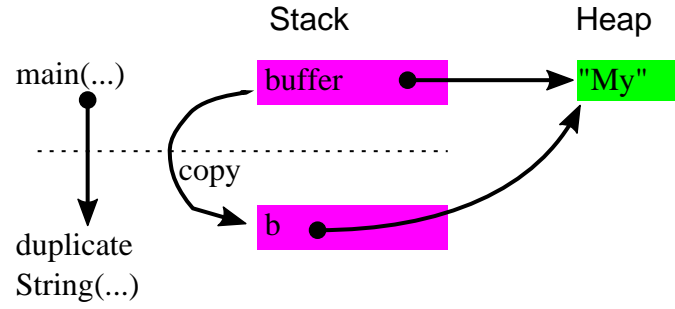
Heap

"My"

Calling duplicateString() method
thereby opening new stack frame.
Copy StringBuffer heap reference
variable buffer into b.

«call-by-reference» details

```
... main(...) {  
  StringBuffer buffer=  
  new StringBuffer("My");  
  ...println(  
  "Before duplicateString:"  
  + buffer);  
  duplicateString(buffer);  
  ...println(  
  "After duplicateString:"  
  + buffer);  
}  
...duplicateString(  
  StringBuffer b){  
  b.append(b);}
```



Using reference **b** for appending "My" to itself. Subsequently returning to main(...) thereby releasing duplicateString's related stack frame (pop() -operation).

«call-by-reference» details

```
... main(...) {  
  StringBuffer buffer=  
    new StringBuffer("My");  
  ...println(  
    "Before duplicateString:"  
    + buffer);  
  duplicateString(buffer);  
  ...println(  
    "After duplicateString:"  
    + buffer);  
}  
...duplicateString(  
  StringBuffer b){  
  b.append(b);}
```

main(...)

Stack

buffer

Heap

"MyMy"

Using reference variable buffer for
printing StringBuffer heap instance.

No «call-by-reference» in Java™!

```
public static void main(String[] args) {
    StringBuffer buffer = new StringBuffer("My");
    System.out.println("Before duplicateString: "
        + buffer);
    replaceString(buffer);
    System.out.println("After duplicateString: "
        + buffer);
}
static void replaceString(StringBuffer b) {
    b = new StringBuffer("Replacement");
}
```

Before duplicateString: *My*
After duplicateString: *My*

No «call-by-reference» details

```
... main(...) {
StringBuffer buffer
= new StringBuffer("My");
...println(
"Before: " + buffer);
replaceString(buffer);
...println(
"After: " + buffer);
}
```

Stack

Heap

main(...)

Create StringBuffer instance on heap.

```
replaceString(StringBuffer b){
b = new StringBuffer(
"Replacement");
}
```

No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```

main(...)

Stack

Heap

"My"

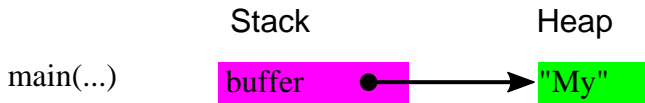
```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```

Create reference variable buffer
initializing with StringBuffer heap
reference.

No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```

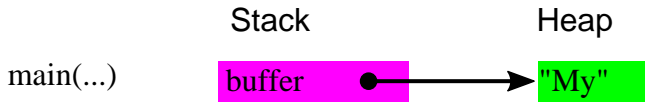
```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```



Using variable buffer's reference
to print heap instance.
Result: "Before: My"

No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```

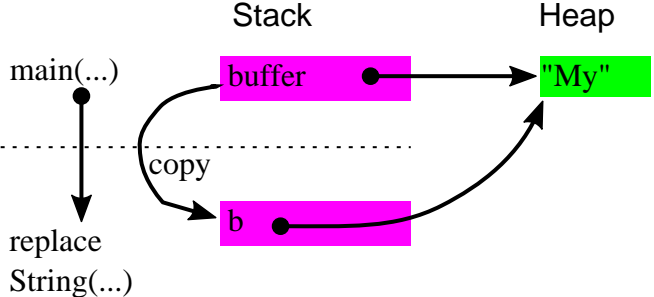


Calling `replaceString()` method.

```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```

No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```



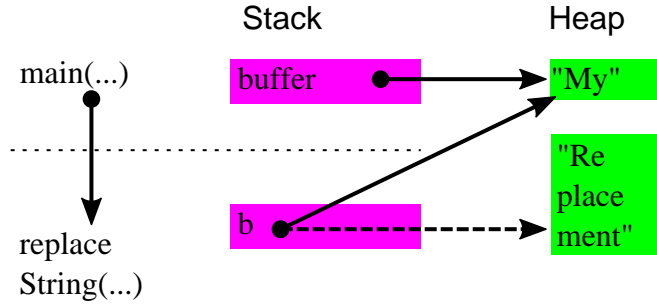
Creating second instance on heap.

```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```


No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```

```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```



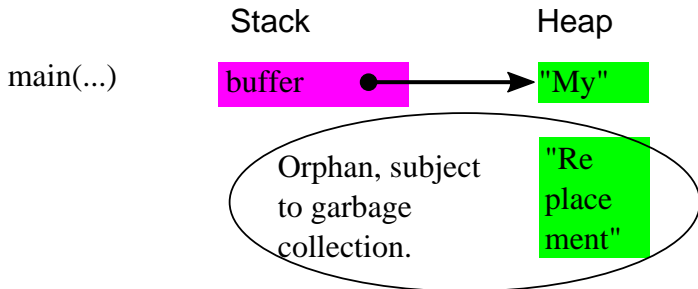
Changing reference to new heap instance.

Return to main(...) thereby releasing stack frame.

No «call-by-reference» details

```
... main(...) {  
  StringBuffer buffer  
  =new StringBuffer("My");  
  ...println(  
    "Before: " + buffer);  
  replaceString(buffer);  
  ...println(  
    "After: " + buffer);  
}
```

```
replaceString(StringBuffer b){  
  b = new StringBuffer(  
    "Replacement");  
}
```



Print original instance. Result:

"After: My"

Reference to second instance died
along with its defining stack frame
leaving an orphaned heap instance
behind.

C++ reference operator "&"

```
int a = 1;
int &b = a;
cout << a << " : " << b << endl;
a = 5;
cout << a << " : " << b << endl;
```

```
1 : 1
5 : 5
```

C++ offers «call-by-reference» by virtue of “&”

```
// Passing a reference
// to variable n
void printDuplicateValue(int &n) {
    n = 2 * n;
    cout << "duplicateValue: " << n
         << endl;
}
int main() {
    int value = 3;
    cout << "Before call: "
         << value << endl;
    printDuplicateValue(value);
    cout << "After call: "
         << value << endl;
}
```

```
Before call: 3
duplicateValue: 6
After call: 6
```

C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
    << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
    << n ...;  
}
```

Stack

main(...)

Create int variable assigning initial
value 3.

C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
  << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
  << n ...;  
}
```

main(...)

Stack

value: 3

Print content of variable value.
Result: 3

C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
  << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
  << n ...;  
}
```

main(...)

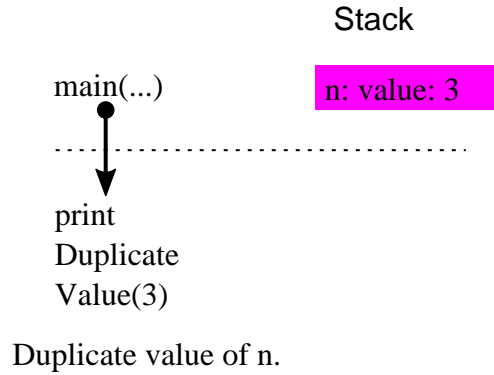
Stack

value: 3

Call method
printDuplicateValue(3)
Opening stack frame. Alternate
variable n addressing identical
memory content.

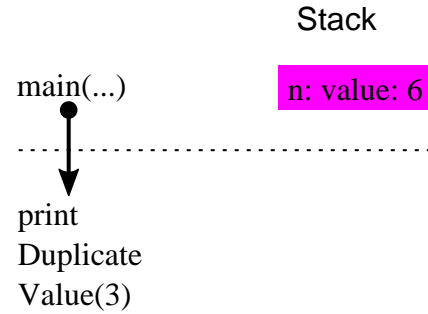
C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
    << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
    << n ...;  
}
```



C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
    << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
  << n ...;  
}
```



Print value of variable n.

Result: 6

Return to main(...) method releasing
stack frame.

C++ «call-by-reference» details

```
... int main() {  
  int value = 3;  
  cout << "Before  
  printDuplicateValue: "  
    << value << endl;  
  printDuplicateValue(value);  
  cout << "Before  
  printDuplicateValue: " ...  
}... printDuplicateValue(  
  int& n){  
  n = 2 * n;  
  ... "printDuplicateValue: "  
    << n ...;  
}
```

main(...)

Stack

value: 6

Print variable value.
Result: 6

Method calling

```
public class Circle {
    static final double
        PI = 3.141592653589793;
    double r;
    /** Change a circle's area
     * @param area The desired new area
     * @return The circle's new radius */
    double setArea(final double area ❶) {
        double val ❷ = area / PI ❸;
        return ❹ r ❺ = Math.sqrt(val);
    }
}
```

❹ returning values.

- ❶ Passing arguments.
- ❷ Defining method local variables.
- ❸ Accessing class variable.

❺ Accessing instance variable.

Three variable scopes

```
public class Circle {  
    static final double  
        PI ❶ = 3.141592653589793;  
  
    double r ❷ ;  
  
    double setArea(final double area ❸) {  
        double val ❸ ...  
        ...  
    }  
}
```

- ❶ Class scope.
- ❷ Instance scope
- ❸ Method scope

Scope lifetimes

Class scope (s t a t i c)

Instance scope

Method scope

Application process

Object lifetime: new (. . .) until being garbage collected.

Method invocation until r e t u r n.

Two runtime memory categories

Heap memory

- Allocation of class or array instances:

```
new String()
```

```
new float [ 200]
```

- De-allocation subject to garbage collection.

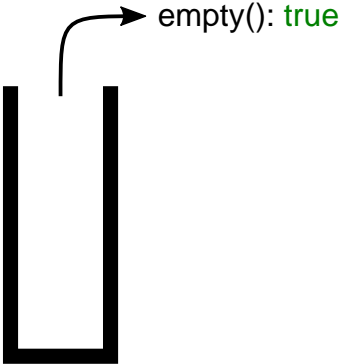
Execution stack

- One instance per process thread.
- Hosting variables (values or references)

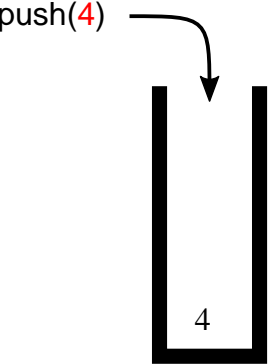
Stack: Four operations

Method	Description	Precondition	Access type
<code>push(. . .)</code>	Add object on top	-	Modify
<code>pop()</code>	Read + remove topmost object	Stack not empty	Modify
<code>top()</code>	Read topmost object	Stack not empty	Read only
<code>empty()</code>	return true if and only if stack is empty	-	Read only

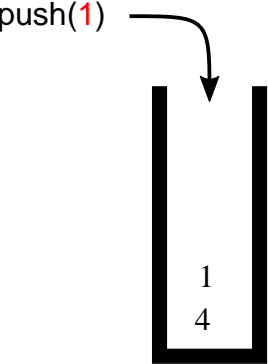
Example: Storing integer values



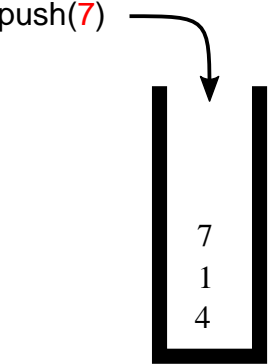
Example: Storing integer values



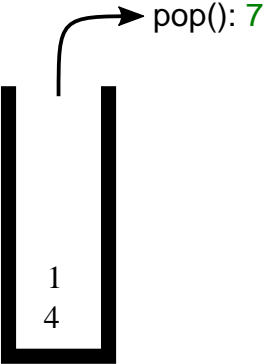
Example: Storing integer values



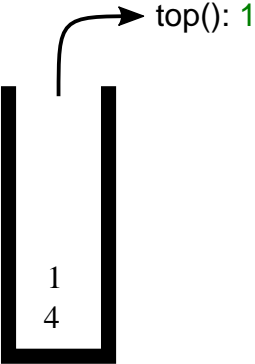
Example: Storing integer values



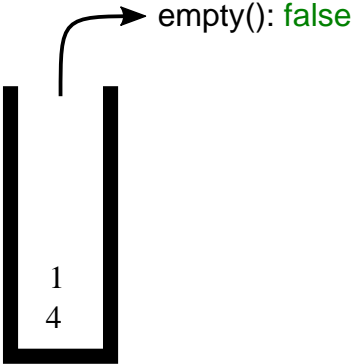
Example: Storing integer values



Example: Storing integer values



Example: Storing integer values



Method calling

```
final Stack<Integer> si =  
    new Stack<>();
```

```
si.push(4);
```

```
si.push(1);
```

```
si.push(7);
```

```
System.out.println("Top: " + si.peek()); // top
```

```
while (! si.empty()) {
```

```
    System.out.println("Not empty: " + si.pop());
```

```
}
```

Create new stack of integer values

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

Add integer value 4 to the stack



Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

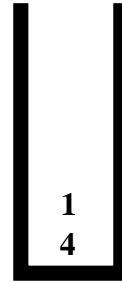
Add integer value 1 to the stack



Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

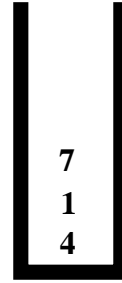
Add integer value 7 to the stack



Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

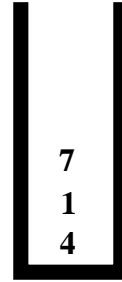
**Read stack's topmost element and
write its value to standard output
(leave stack untouched)**



Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

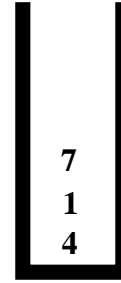
**Check if stack is empty
(not yet)**



Top: 7

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```



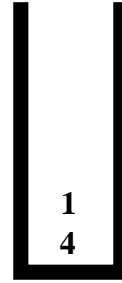
Remove **7** from stack and write its value to standard output.

Top: 7

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

**Check if stack is empty
(not yet)**

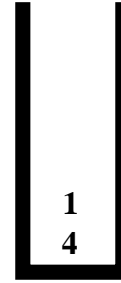


Top: 7
Not empty: 7

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

Remove **1** from stack and write its value to standard output.



Top: 7
Not empty: 7

Method calling

```
final Stack<Integer> si =
    new Stack<>();
si.push(4);
si.push(1);
si.push(7);
System.out.println("Top: " + si.peek()); // top
while (! si.empty()) {
    System.out.println("Not empty: " + si.pop());
}
```

**Check if stack is empty
(not yet)**



Top: 7
Not empty: 7
Not empty: 1

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

Remove 4 from stack and write its value to standard output.

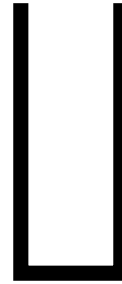


```
Top: 7  
Not empty: 7  
Not empty: 1
```

Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

**Check if stack is empty: true
(Program termination)**



Top: 7
Not empty: 7
Not empty: 1
Not empty: 4

Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



Call stack trace

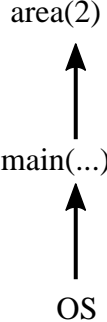
```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



args: ...

Call stack trace

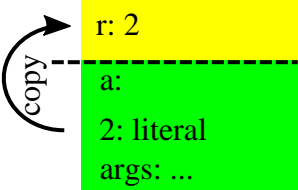
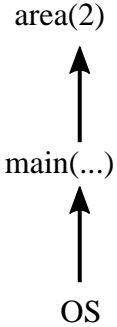
```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



a:
2: literal
args: ...

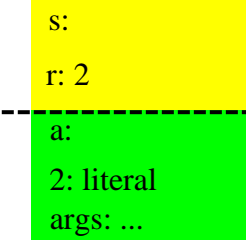
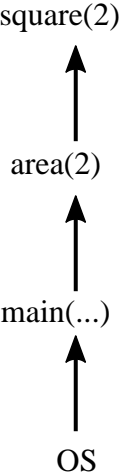
Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



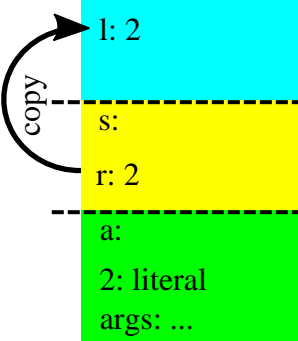
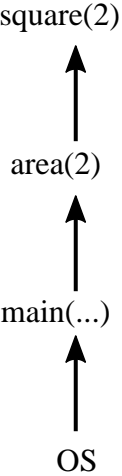
Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



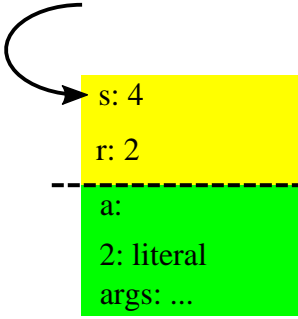
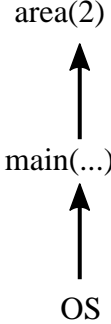
Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



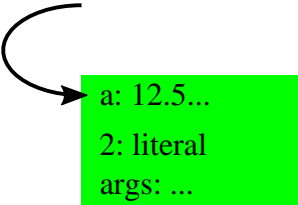
Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```



Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args){  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```

OS

IDE debugger

```
2  @  static double square(double l) {  l: 2.0
3  |  return l * l;  l: 2.0
4  |  }
5  @  static double circleArea(double r) {
6  |  final double s = square(r);
7  |  return Math.PI * s;
8  |  }
9  ▶  public static void main(String[] args) {
10 |  double a = circleArea(2);
11 |  System.out.println("Area:" + a);
12 |  }
```

CallStackExample > square()

Debug CallStackExample (1)

Debugger Console [Icons]

Frames Variables

1	"main"@1 in group "main..."	
	square:3, CallStackExample	
	circleArea:6, CallStackExample	

2	l = 2.0
---	---------

Motivation

- Modeling finite sets of discrete states.

- Examples:

A room's door: {OPEN, CLOSED}

State of matter: {SOLID, LIQUID, GASEOUS}

- No dynamic change of state set.

Weekly offered lectures

```
public class Lecture {  
  
    public final int dayHeld; /* e.g. to be held on Tuesdays */  
  
    public final String title; /* e.g. «PHP introduction» */  
  
    public Lecture(final int dayHeld, final String title) {  
        this.dayHeld = dayHeld;  
        this.title = title;  
    }  
}
```

Weekly offered lectures by simple numbers

Quick and dirty:

Class Driver:

final Lecture

```
    phpIntro      = new Lecture(1 /* Monday */, "PHP introduction"),  
    advancedJava = new Lecture(5 /* Friday */, "Advanced Java");
```

Error prone:

- Weeks start on Mondays?
- Index starts with 0 or 1?

Weekdays i nt representation

```
public class Day {  
  
    static public final int  
        MONDAY    = 1,  
        TUESDAY   = 2,  
        WEDNESDAY = 3,  
        THURSDAY  = 4,  
        FRI DAY   = 5,  
        SATURDAY  = 6,  
        SUNDAY    = 7;  
}
```


Weekly offered lectures using constants

Class Driver:

final Lecture

```
phpIntro = new Lecture( Day. MONDAY, "PHP i nt roduct i on" ),  
advancedJava = new Lecture( Day. FRI DAY, "Advanced Java" );
```

Converting index values to day names

```
public class Day {  
    ...  
    public static String getDaysName(final int day) {  
        switch (day) {  
            case MONDAY:    return "Monday";  
            case TUESDAY:   return "Tuesday";  
            case WEDNESDAY: return "Wednesday";  
            case THURSDAY:  return "Thursday";  
            case FRIDAY:    return "Friday";  
            case SATURDAY:  return "Saturday";  
            case SUNDAY:    return "Sunday";  
  
            default:       return "Illegal day's code: " + day;  
        }  
    }  
}
```

Providing lecture info

```
public class Lecture {
    public final int dayHeld;
    ...

    public String toString() {
        return "Lecture «" + title + "» being held each " +
            Day.getDaysName(dayHeld);
    }
}
```

Sample lectures

```
// Class Driver
final Lecture

    phpIntro = new Lecture(
        Day.MONDAY, "PHP i n t r o d u c t i o n" ),

    advancedJava = new Lecture(
        Day.FRI DAY, "Advanced Java" );

System.out.println(phpIntro);
System.out.println(advancedJava);
```

```
Lecture «PHP i n t r o d u c t i o n»
    being held each Monday
Lecture «Advanced Java»
    being held each Friday
```

Bogus index value

```
// Class Screwed
```

```
final Lecture phpIntro =  
    new Lecture(88, "PHP introduction");
```

```
System.out.println(phpIntro);
```

Lecture «PHP introduction» being
held each Illegal day's code: 88

Pitfall: Method argument order mismatch

```
/**
 * Charge double prices on weekends
 * @param day Day of week
 * @param amount
 * @return the effective amount for
 *         given day of week.
 */
static public int getPrice(
    final int day, final int amount) {
    switch (day) {
        case Day.SATURDAY:
        case Day.SUNDAY: return 2 * amount;

        default: return amount;
    }
}
```

```
// Correct
System.out.println(
    getPrice(Day.SUNDAY, 2));

// Argument mismatch
System.out.println(
    getPrice(2, Day.SUNDAY));

4
7
```

Bad: No warning message whatsoever!

Enumeration by class instances

Roadmap:

- Define a dedicated enumeration representing class.
- Create exactly one class instance per enumeration value.
- Enumeration value equality comparison by virtue of the `==` operator.

Class instance per enumeration value

```
public class Day {  
  
    static public final Day  
        MONDAY    = new Day(),  
        TUESDAY   = new Day(),  
        WEDNESDAY = new Day(),  
        THURSDAY  = new Day(),  
        FRI DAY   = new Day(),  
        SATURDAY  = new Day(),  
        SUNDAY    = new Day();  
}
```

Note: Class without instance attributes.

switch no longer works

Reverting to if .. else if ...

```
public static String getDaysName(final Day day) {
    if (MONDAY == day) { // Switch no longer possible, sigh!
        return "Monday";
    } else if (TUESDAY == day) {
        ...
    } else if (SUNDAY == day) {
        return "Sunday";
    } else {
        return "Illegal day instance: " + day;
    }
}
```

Re-writing get Pri ce()

```
/**
 * Charge double prices on weekends
 * @param day Day of week
 * @param amount
 * @return the effective amount depending on day of week.
 */
static public int getPrice(final Day day, final int amount) {
    if (Day.SATURDAY == day || Day.SUNDAY == day) {
        return 2 * amount;
    } else {
        return amount;
    }
}
```

Compile time argument mismatch error

Preventing method argument order mismatch:

```
// Class Driver
```

```
// o.k.
```

```
System.out.println(Screwed2.getPrice(Day.SUNDAY, 2));
```

```
// Argument mismatch causing compile time type violation error
```

```
System.out.println(Screwed2.getPrice(2, Day.SUNDAY));
```

Pitfall: Creating an undesired instance

Class Screwed:

```
final Day PAST_SUNDAY = new Day();
```

```
final Lecture phpIntro = new Lecture(  
    PAST_SUNDAY, "PHP introduction");
```

```
System.out.println(phpIntro.toString());
```

Lecture «PHP introduction» being
held each **Illegal day instance:**
de.hdm_stuttgart.mi.sd1.
class_wrapper.Day@63961c42

Define a private Day constructor

```
public class Day {  
  
    // Disallow object creation outside class  
  
    private Day() {}  
  
    static public final Day  
        MONDAY    = new Day(),  
        TUESDAY   = new Day(),  
        ...  
        SUNDAY    = new Day();  
}
```

Preventing undesired Day instance creation

Class Screwed:

```
Day PAST_SUNDAY = new Day();
```

```
Lecture phpIntro = new Lecture(  
    PAST_SUNDAY, "PHP introduction");
```

```
System.out.println(phpIntro.toString());
```

Compile time error:

```
'Day()' has private access in  
'de.hdmi.stuttgart.mi.sd1.  
class_wrapper_private.Day'
```

Adding a day name attribute

```
public class Day {
    public final String name;

    private Day(final String name) {
        this.name = name;
    }
    static public final Day
        MONDAY    = new Day("Monday"),
        ...
        SUNDAY    = new Day("Sunday");

    public String toString() { return name; }
}
```

enumDay replacing public class Day

```
public enum Day {  
  
    MONDAY("Monday"),  
    TUESDAY("Tuesday"),  
    ...  
    SUNDAY("Sunday");  
  
    final String name;  
    Day(final String name) { this.name = name;}  
  
    public String toString() { return name;}  
}
```


switch statements working again

```
public enum Day {  
    ...  
    public static String getItalianDayName(final Day day) {  
        switch (day) {  
            case MONDAY:    return "Lunedì";  
            case TUESDAY:   return "Martedì";  
            ...  
            case SUNDAY:    return "Domenica";  
        }  
        return null; // Actually unreachable, but static  
                    // compiler code analysis is limited  
    }  
}
```

enumconstructor being implicitly private

```
public enum Day {  
    ...  
    private Day(final String name)  
    { ...
```

```
public enum Day {  
    ...  
    public Day(final String name)  
    { ...
```

Prohibits enumexternal instance creation.

Compile time warning:

Modifier 'private' is redundant for enumconstructors

Compile time error:

Modifier 'public' not allowed here

Related exercises

Exercise 123: Compass directions

Exercise 124: Compass direction neighbours

Git:

1. A completely ignorant, childish person with no manners.
2. A person who feels justified in their callow behaviour.
3. A pubescent kid who thinks it's totally cool to act like a moron on the internet, only because no one can actually reach through the screen and punch their lights out.

Useful links

- [A recipe for disaster](#)
- [Git tutorial](#)
- [A Practical Introduction to git](#)

Initialize git project

- `git init`
- Result: Sub folder `.git` containing project's metadata and versioning history

Configure author related data.

- `git config --global user.email "foo@company.com"`
- `git config --global user.name "Helen Foo"`

Adding resources to project index and staging area

- `public class Math {`

```
    static public int add(  
        final int a, final int b) {  
        return a + b;  
    }  
}
```

- `git status`: Math.java yet unversioned.

- `git add Math.java`

- `git status`: Math.java became versioned.

Committing change set

- `git commit -m "New math class containing single method"`
- Result: Commit change set to repository adding given message to project log.

Project versioning status

git status

On branch master ❶

No commits yet ❷

Untracked files: ❸

(use "git add <file>..." to include in what will be committed)

Math.java

nothing added to commit but untracked files present (use "git add" to track) ❹

Adding a comment

```
public class Math {  
  
    static public int add(  
        final int a, final int b) {  
        return a + b;  
    }  
}
```

```
public class Math {  
    /**  
     * Summing two int values.  
     * @param a first value.  
     * @param b second value.  
     * @return The sum of both.  
     */  
    static public int add(  
        final int a, final int b) {  
        return a + b;  
    }  
}
```

git diff tracing changes

```
index 5d72bde..d273b77 100644
--- a/Math.java
+++ b/Math.java
@@ -1,5 +1,10 @@
public class Math {
+
+ /**
+  * Summing two int values.
+  * @param a first value.
+  * @param b second value.
+  * @return The sum of both.
+  */
static public int add(
final int a, final int b) {
return a + b;
}
```

Reverting individual file.

```
>git checkout -- ❶ Math.java ❷
```

```
>git diff Math.java ❸
```

```
>
```

❷ Replace working tree file Math.java by repository master.

❶ Double dashes '--' disambiguating branch names from file names.

❸ No difference working tree to master branch.

Compiling, Math.class and Print.class.

```
> javac Print.java Math.java    Compilation creating Math.class and Print.class
```

```
> git status
```

```
On branch master
```

```
Your branch is up to date with 'origin/master'.
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
Math.class
```

```
Print.class
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

Math.class, Print.class and versioning.

1. Math.class and Print.class are being generated from Math.java and Print.java.
2. Rule of thumb: Do not version dependent objects.

Solution: Add a .gitignore file to versioning to contain:

```
# ignore generated .class files  
*.class
```

Show project's log

git log

commit 0137ccd857a242f4751e36bdbce365c6130c3a32 ① (HEAD -> master)

Author: Martin Goik <goik@hdm-stuttgart.de>

Date: Sat May 25 11:56:00 2019 +0200

Removing duplicate headline ②

commit 7f119fac36e02e4c5a7f04f022217b6f744d6e1d ③

Author: Martin Goik <goik@hdm-stuttgart.de>

Date: Sat May 25 11:49:52 2019 +0200

Project Readme.md ④ ...

Switch to an older revision ...

```
git checkout 7f119fac36e02e4c5a7f04f022217b6f744d6e1d
```

```
Note: checking out '7f119fac36e02e4c5a7f04f022217b6f744d6e1d'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 7f119fa Project README.md
```

... and forth to current master's HEAD

```
git checkout master
```

```
Previous HEAD position was 7f119fa Project Readme.md
```

```
Switched to branch 'master'
```

Related exercises

Exercise 125: git local, DIY

Centralized remote repository

See multi+remote/shared git:

1. Create empty remote repository on `gitlab.mihdm-stuttgart.de`.
2. Connect local repository to remote
3. push or pull content

Step 1: Create remote repository

← → ↻ 🔒 https://gitlab.mi.hdm-stuttgart.de/projects/new ☆

GitLab Projects ▾ Groups ▾ Activity Milestones Snippets 🔔 + ▾ Search or jump to... 🔍

new (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

To only use CI/CD features for an external repository, choose **CI/CD for external repo**.

Tip: You can also create a project from the command line. [Show](#)

1

Project name

GitIntro

Project URL

https://gitlab.mi.hdm-s goik ▾

Project slug

gitintro

Want to house several dependent projects under the same namespace? [Create a group](#)

Project description (optional)

Description format

2

Step 2: Retrieve remote repository address



GitIntro 
Project ID: 2764

[Add license](#)

The repository for this project is empty

You can create files directly in GitLab using one of the following options.

[New file](#)

[Add README](#)

[Add CHANGELOG](#)

[Add CONTRIBUTING](#)



Star

0

Clone with SSH

```
git@gitlab.mi.hdm-stuttgart.
```

Clone with HTTPS

```
https://gitlab.mi.hdm-stuttg
```

Cop

Command line instructions

You can also upload existing files from your computer using the instructions below.

Git global setup

Step 2: Connect to remote repository

```
git remote add origin ① https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
```

① **origin** is an alias for our remote repository `https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git`.

Step 3: Push local to remote

```
git push --set-upstream origin ❶ master
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik ❷
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
Counting objects: 5, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 507 bytes | 507.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
 * [new branch]      master -> master ❸
Branch 'master' set up to track remote branch 'master' from 'origin'.
```


Step 3: Pull remote to local

```
> git pull ❶
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik ❷
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From https://gitlab.mi.hdm-stuttgart.de/goik/gitintro ❸
   733e541..ffff092  master    -> origin/master
Updating 733e541..ffff092 ❹
Fast-forward
 Math.java | 10 ++++++-- ❺
 1 file changed, 8 insertions(+), 2 deletions(-)
```

Alternative: Create remote, then clone

1. Create new possibly non-empty project at <https://gitlab.mi.hdm-stuttgart.de>.
2.

```
> git clone https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
Cloning into 'gitintro'...
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
```

Conflicting changes

User A: Print.java

```
// Driver class
public class Print {
    public static void
        main(String[] args) {
        System.out.println
            (Math.add(2, 3));
    }
}
```

User B: Print.java

```
/**
 * Application entry point
 */
public class Print {
    public static void
        main(String[] args) {
        System.out.println
            (Math.add(2, 3));
    }
}
```

Commit schedule

User A

Edit: ... **Driver class** ...

git commit, git push

-

-

-

User B

-

-

edit: ... **Application entry point** ...

git commit

git push: **Fail!**

User B: git push fails

```
>git push ...
```

```
To https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
```

```
! [rejected]      master -> master (fetch first)
```

```
error: failed to push some refs to
```

```
  'https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git'
```

```
hint: Updates were rejected because the remote contains work that you do
```

```
hint: not have locally. This is usually caused by another repository pushing
```

```
hint: to the same ref. You may want to first integrate the remote changes
```

```
hint: (e.g., 'git pull ...') before pushing again.
```

User B: git pull fails as well

```
>git pull
```

```
...
```

```
From https://gitlab.mhdm-stuttgart.de/goik/gitintro
```

```
  b003a82..dbbedb0  master    -> origin/master
```

```
Auto-merging Print.java
```

```
CONFLICT (content): Merge conflict in Print.java
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflict details

```
>git diff Print.java
diff --cc Print.java
index fc36ae6,7b27edf..0000000
--- a/Print.java
+++ b/Print.java
@@ -1,6 -1,4 +1,10 @@
++<<<<<< HEAD
+/**                               ❶
+ * Application entry point
+ */
++=====
+ // Driver class ❷
++>>>>>>> dbbedb0fc29d77beeaada37f2538d78f82bac93
public class Print {
    public static void
        main(String[] args) {
            System.out.println
                (Math.add(2, 3));
        }
}
```

Struggling for resolution



Merging Print.java manually

```
<<<<<<< HEAD
/**
 * Application entry point
 */
=====
// Driver class
>>>>>>> 10cf21c ... 759462c
public class Print {
    public static void
        main(String[] args) {
        System.out.println
            (Math.add(2, 3));
    }
}
```



```
/**
 * Driver class, application entry point
 */
public class Print {
    public static void
        main(String[] args) {
        System.out.println
            (Math.add(2, 3));
    }
}
```

Commit and push merge

```
>git add Print.java
```

```
>git commit --message "Merging changes"  
[master 173487a] Merging changes
```

```
>git push
```

```
...
```

```
To https://gitlab.mhdm-stuttgart.de/golk/gitintro.git  
10cf21c..173487a master -> master
```

Related exercises

Exercise 126: git distributed, DIY