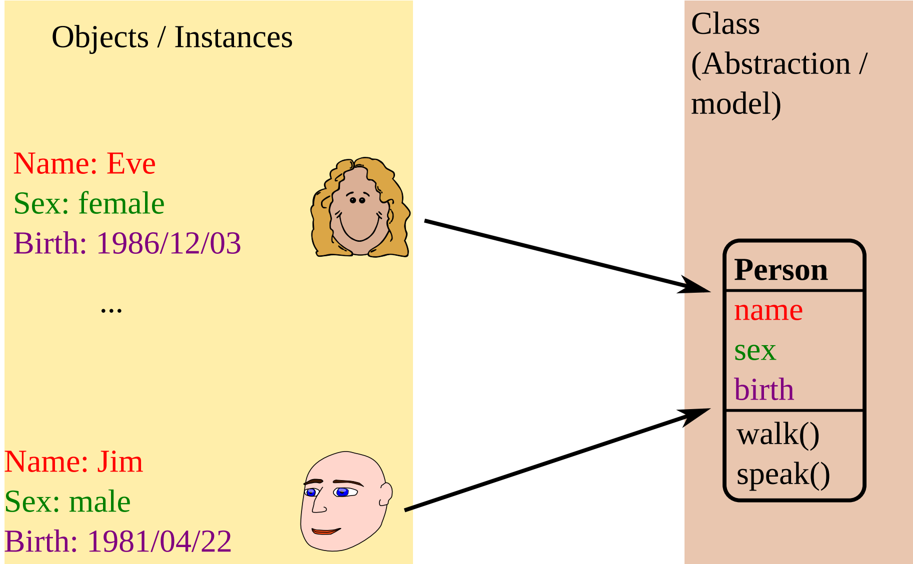
Instances of a Class



General class structure

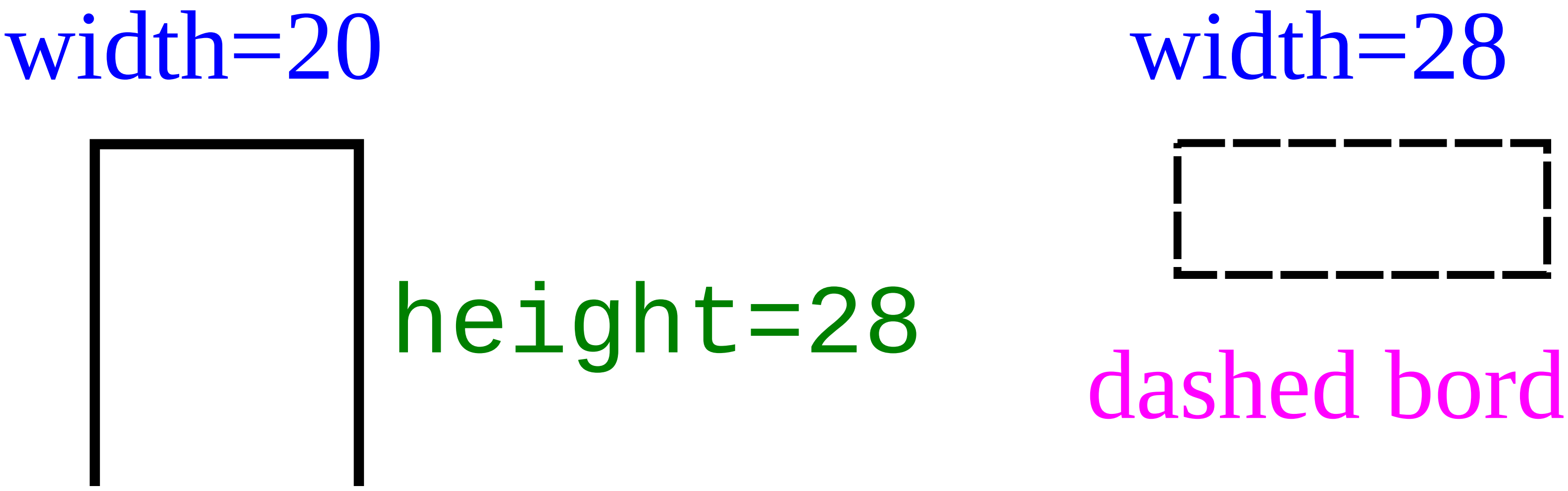
Person	The classes name
name sex birth	Attributes
walk() speak()	Methods

What's a class anyway?

In object oriented languages **classes**:

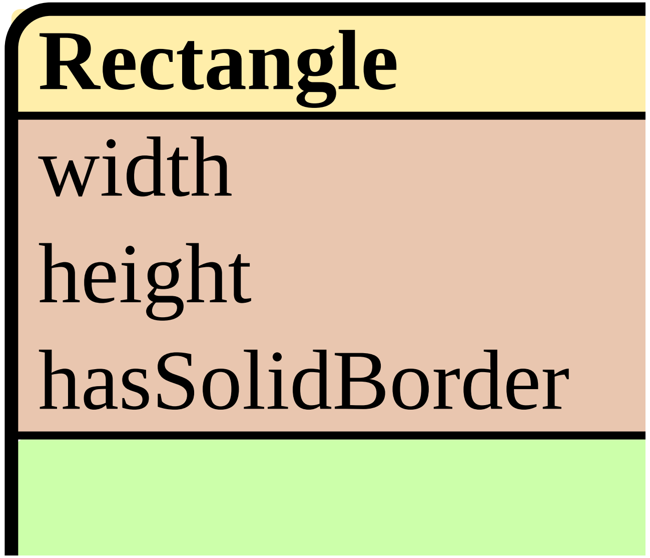
- are blueprints for objects.
- contain attributes and methods.
- allow for implementation hiding.
- allow for tailored access to methods and attributes.

Rectangle objects



A class describing rectangles

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

  // solid or dashed:
  boolean hasSolidBorder;
}
```



Rectangle class and instances

Class

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

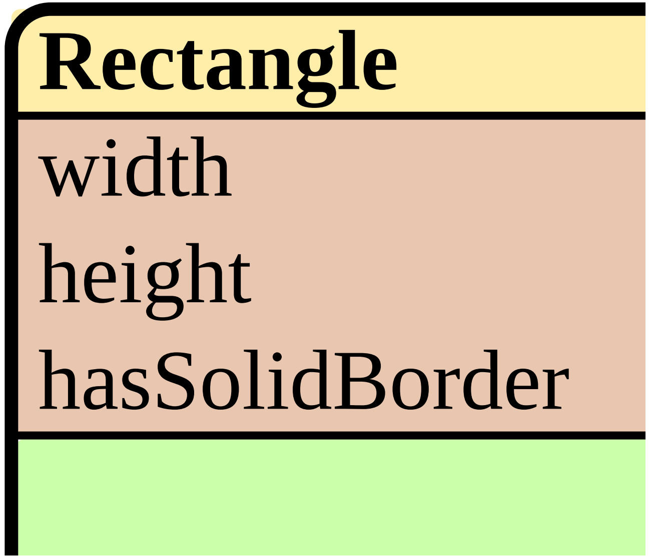
Rectangle class and instances

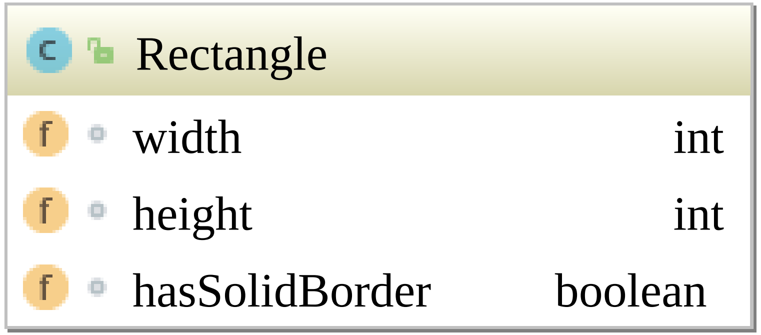
Rectangle class and instances

Rectangle class and instances

Rectangle class and instances

Generated diagrams





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

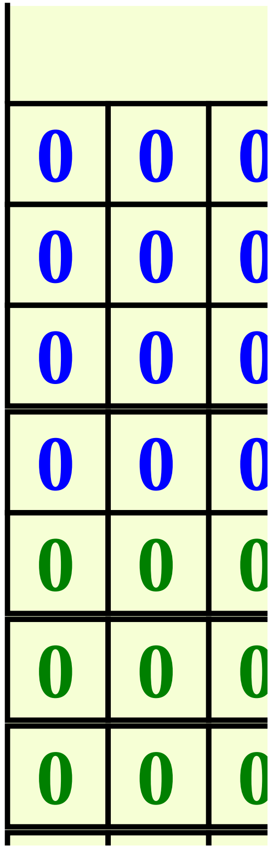
```
Rectangle dashedRectangle = new Rectangle();

dashedRectangle.width = 28;
dashedRectangle.height = 10;
dashedRectangle.hasSolidBorder = false;
```

Syntax accessing object attributes:

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

Instance memory representation

```
Rectangle r =
  new Rectangle();
r.width = 20;
r.height = 30;
r.hasSolidBorder = true;
```



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.hdm_stuttgart.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:

  `public`, `private` and `protected` modifier

- Resolving class name clashes. Example:

  java.lang.String vs. **my.personal**.`String`

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-stuttgart.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();
  }
}
```

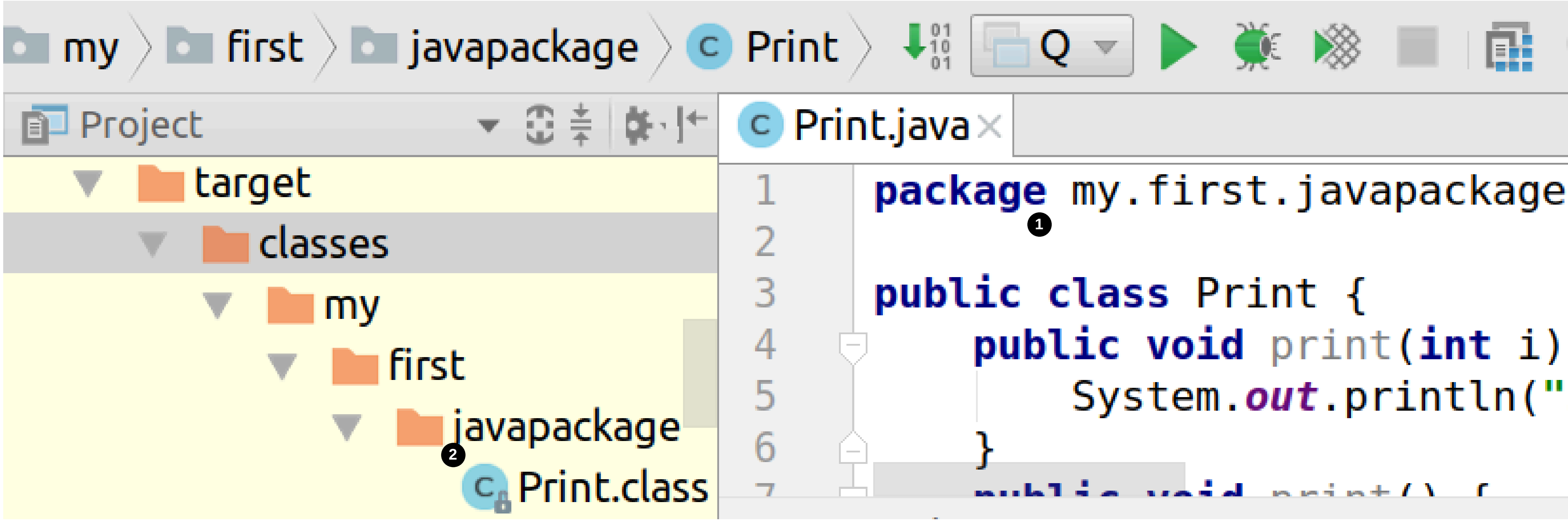
Good

```
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();
  }
}
```

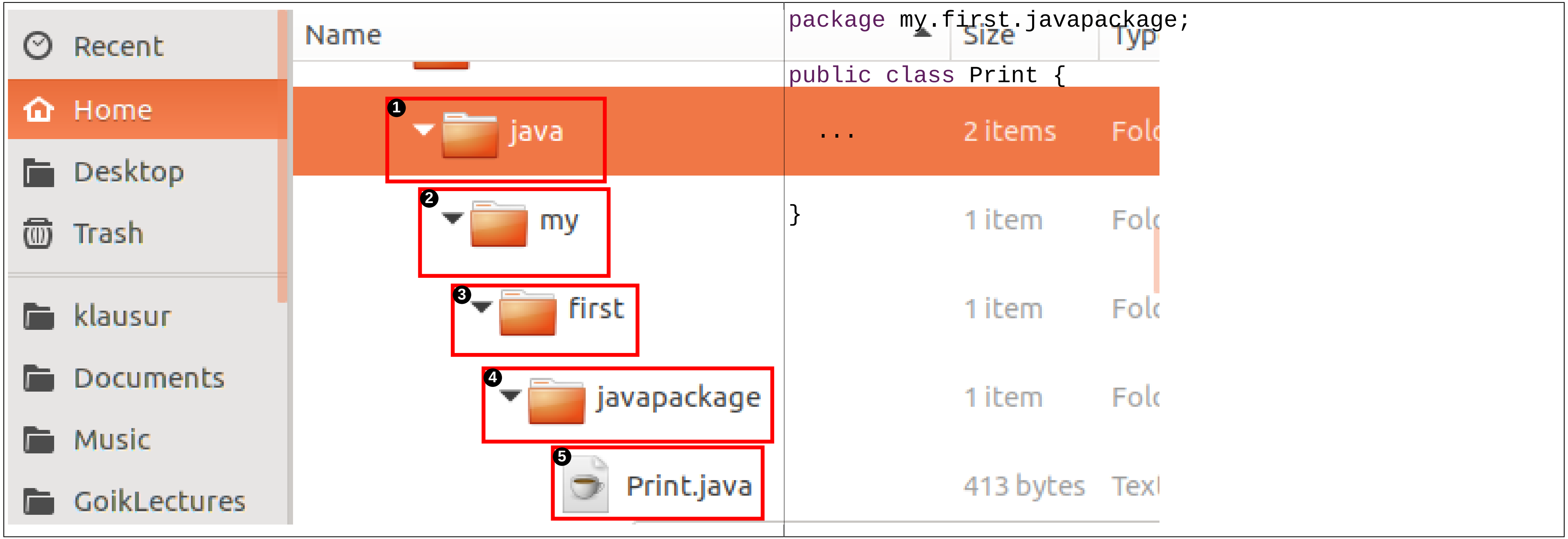
Special: Classes in package java.lang

```
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



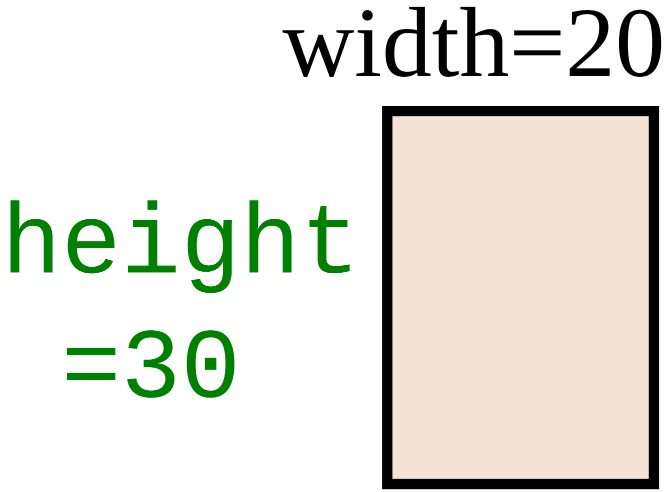
Source hierarchy view



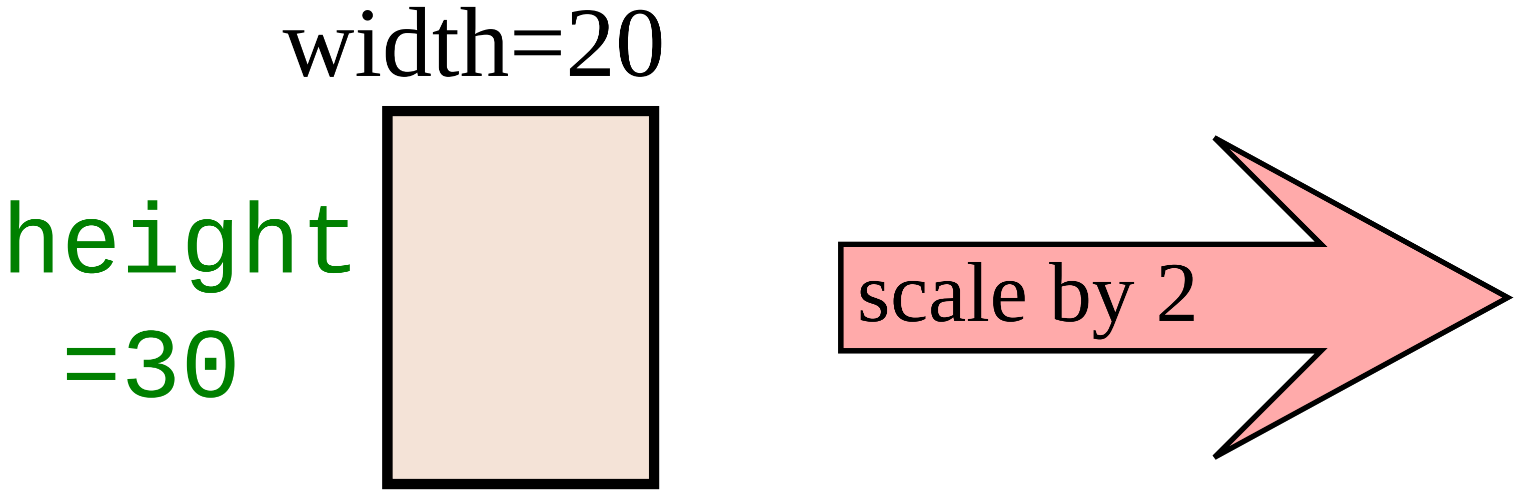
Object methods

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.

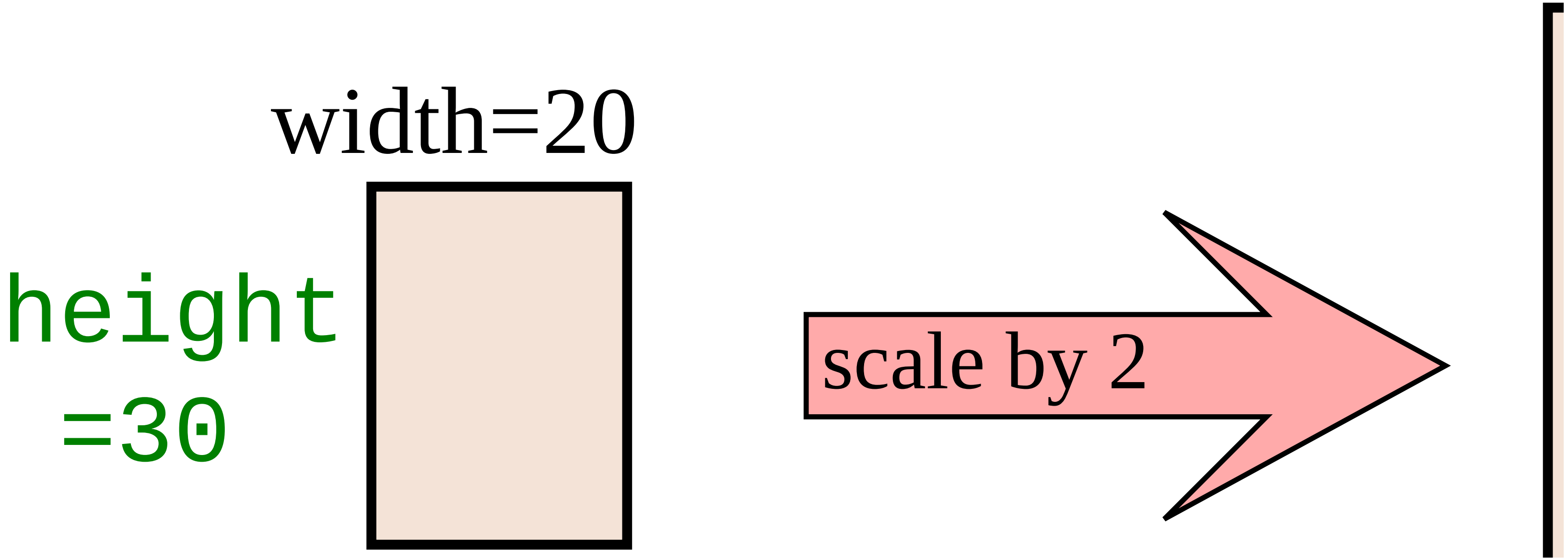
Scaling a rectangle



Scaling a rectangle

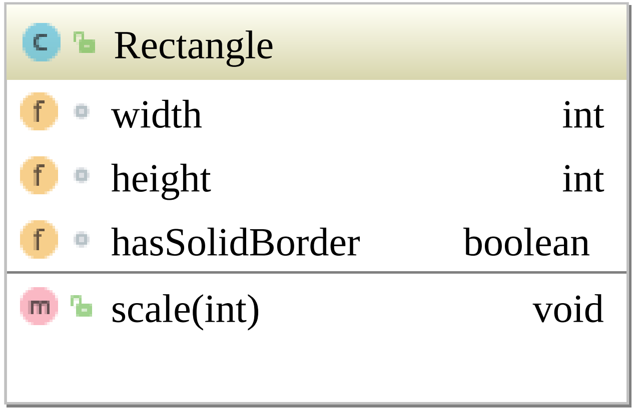


Scaling a rectangle



Scaling method implementation

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

    public void scale (int factor) {
        width *= factor;
        height *= factor;
    }
}
```



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

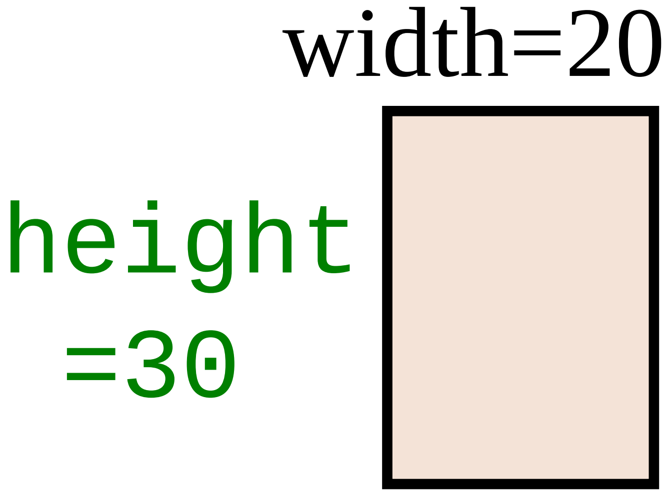
`Rectangle r = new Rectangle();`
`r.width = 33;`
`r.height = 22;`
 	
`r.scale(2);`
 	
`System.out.println("width=" + r.width);`
`System.out.println("height=" + r.height);`
	`width=66`
	`height=44`

Method definition syntax

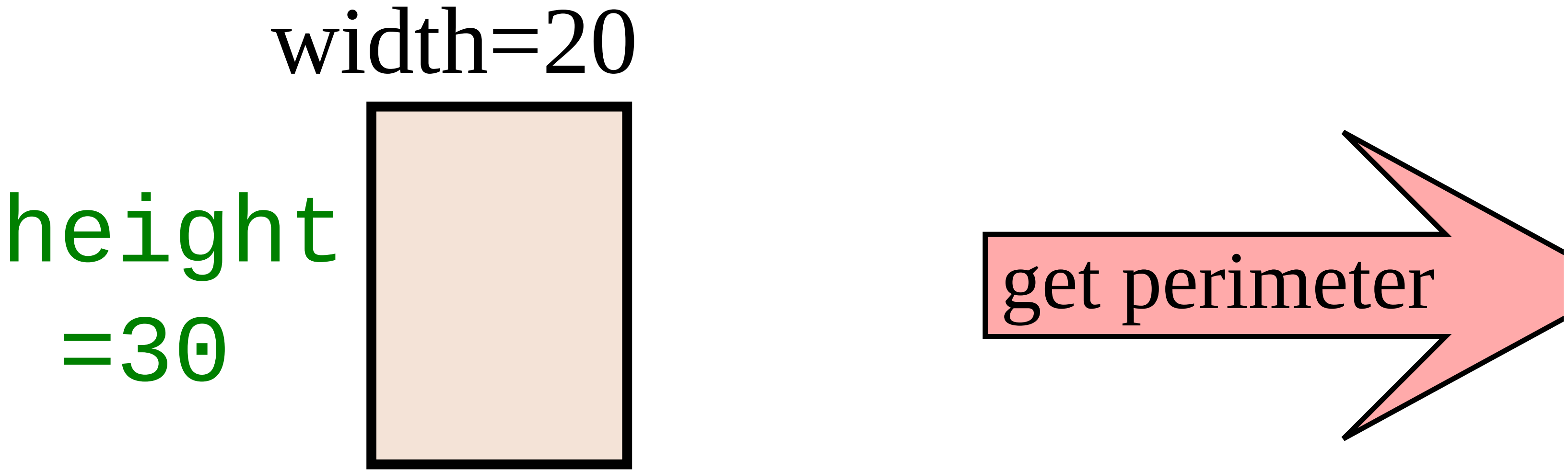
```
public ❶ void ❷ scale❸ (int factor ❹) { ❺
   width *= factor; ❻
   height *= factor;
}
```

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

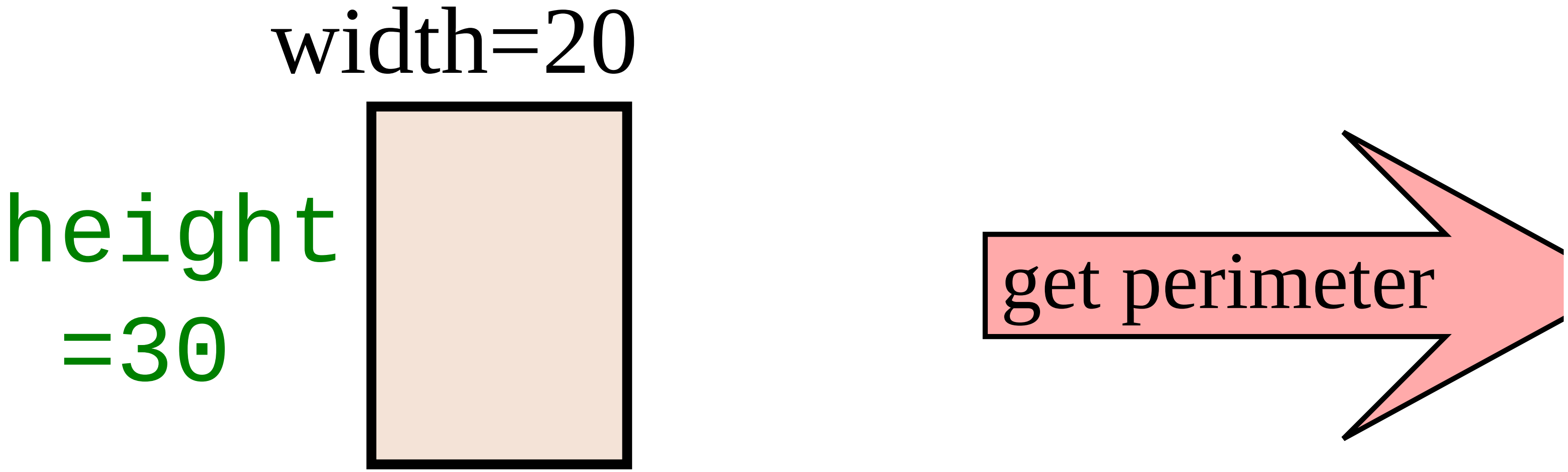
A rectangle's perimeter



A rectangle's perimeter

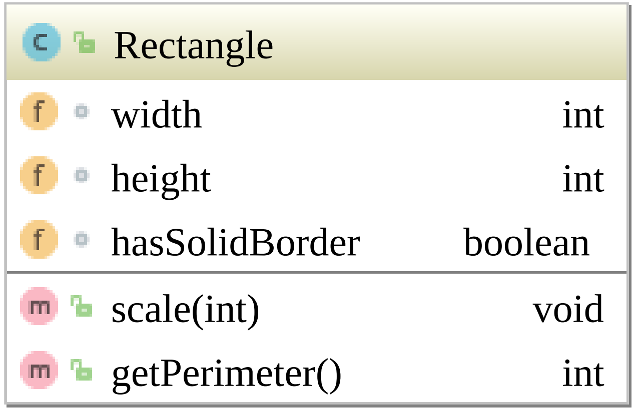


A rectangle's perimeter



`getPerimeter()` method implementation

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

    public void scale (int factor) { ... }

    public int getPerimeter() {
        return 2 * (width + height);
    }
}
```



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

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

Time time = new Time();

time.hour = 17;
time.minute = 45;
```

Setter access

```java
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	Setter / getter access

<pre>public class Time {
  // Minutes since 00:00
  public int minute;
}
 	
final Time time = new Time();
time.minute = 17 * 60 + 45;</pre>
	<pre>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);</pre>

Related exercises

Exercise 87: Implementing getter methods

Defining type signatures

```
boolean ❶ startsWith(String prefix, int toffset) 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
`void print()`	void	(void)
`int add (int a, int b)`	int	(int, int)
`int max (int a, int b)`	int	(int, int)
`void print (int a, float b)`	void	(int, float)
`void display (float a, int b)`	void	(float, int)

Defining method signatures

```
boolean startsWith❶(String prefix, int toffset) ❷
```

❶ Method name `startsWith`.

❷ Number of arguments among with their respective types and order.

Method signature examples

Method	Method name	Method signature
`void print()`	print	(void)
`int add (int a, int b)`	add	(int, int)
`int max (int a, int b)`	max	(int, int)
`void print (int a, float b)`	print	(int, float)
`void display(float a, int b)`	display	(float, int)

Related exercises

Exercise 88: Method signature variants

Method overloading: Same name, different signature

```java
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);
  }
}
```

```java
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

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

<table>
<tr><td rowspan="2">Return type</td><td colspan="2">Method signature</td></tr>
<tr><td>Method name</td><td>Argument type list</td></tr>
<tr><td>String</td><td>getDetails</td><td>(void)</td></tr>
<tr><td>int</td><td>getDetails</td><td>(void)</td></tr>
</table>

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

`int a;`
`a = 33;`
	`Rectangle r = new Rectangle();`
	
	`r.width = 28;`
	`r.height = 10;`
	`r.hasSolidBorder = false;`

Combining statements desired:	
`int a = 33; // works!`	`Rectangle r = new Rectangle(28, 10, false);//how ???`

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");`

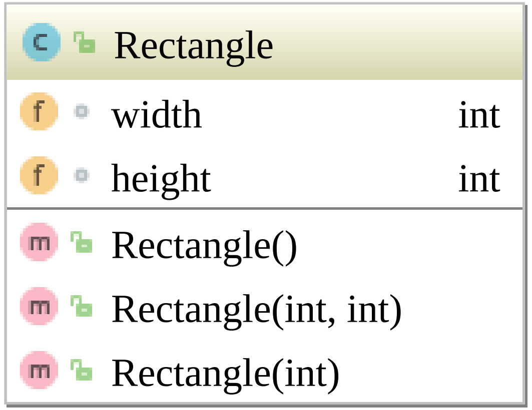
Constructors

- 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

```java
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;
    }
}
```

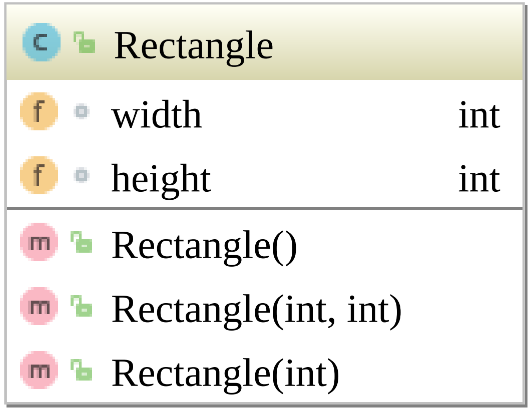


Constructor calls within constructor

<pre>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;
  }
}</pre>
	<pre>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);
	  }
	}</pre>

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
```



No constructor vs. default constructor

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

<pre>public class Rectangle {
    int width, height;
    boolean hasSolidBorder;
 	
    // Default constructor, empty body.
    public Rectangle ( ){}
}</pre>
	<pre>public class Rectangle {
	    int width, height;
	    boolean hasSolidBorder;
	
	  // No constructor at all
	}</pre>

Non - default constructor, but no default constructor

<pre>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.
}</pre>
	<pre>// o.K.: Using non-default
	// constructor.
	
	Rectangle r =
	   new Rectangle(3, 6, false);</pre><hr><pre>// Wrong: Default constructor
	// undefined, but non-default
	// constructor present.
	
	Rectangle r = new Rectangle();</pre>

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;
  }
}
```

Generated Javadoc

Constructor Detail

Circle

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

Creating a circle.

Parameters:

`r` - representing the circle's radius

Generated Javadoc

Constructor Detail

Circle

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

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

Creating a circle.

Parameters:

`r` - representing the circle's radius

Generated Javadoc

Constructor Detail

Circle

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

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

Creating a circle.

Choice requires supplementary explanation

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

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

```java
/** 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?

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

```java
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**

<pre>public class Helper { ❶
 	
  static ❷ public int maximum(int a, int b)
   {
    if (a < b) {
      return b;
    } else {
      return a;
    }
  }
}</pre>
	<pre>// No instance required any longer
	System.out.println("Maximum: " +
	     Helper.maximum(-3, 5)); ❶</pre>

	<pre>Maximum: 5</pre>

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;
  }

}
```

$\boxed{0}$

Once per class

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

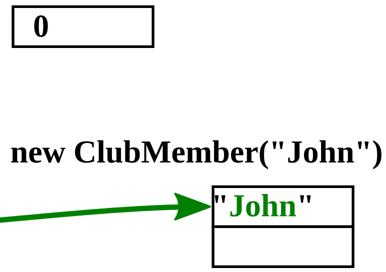
```
public class ClubMember {

  static  private int memberCount = 0;

  final private int memberNumber;
  final private String name;

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

}
```

$\boxed{0}$

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

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")

"John"

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")

"John"

1

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")

"John"
1

new ClubMember("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;
  }

}
```

1

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;
  }

}
```

2

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;
  }

}
```

2

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.print(ln)`

```
   ❶    ❷     ❸
System.out.print(...)
```

❶ Class `System` in package java.lang.

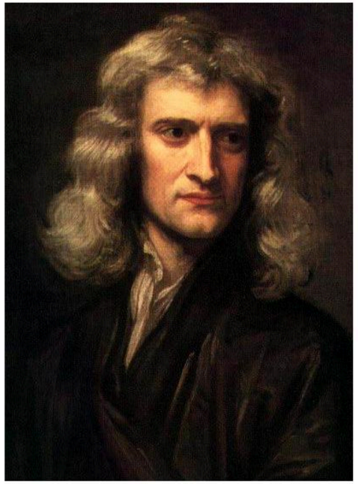
❷ Class variable (static) out of type `PrintStream` in class `System`

❸ 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

Userspace tools

glibc, bash, systemd, ...

uses

Linux kernel + modules

~28,000,000 lines of code

Application execution prerequisites

Java Virtual machine + libraries

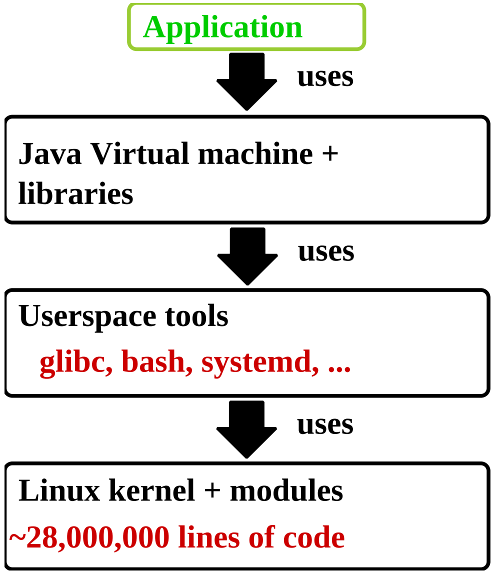
↓ **uses**

Userspace tools
glibc, bash, systemd, ...

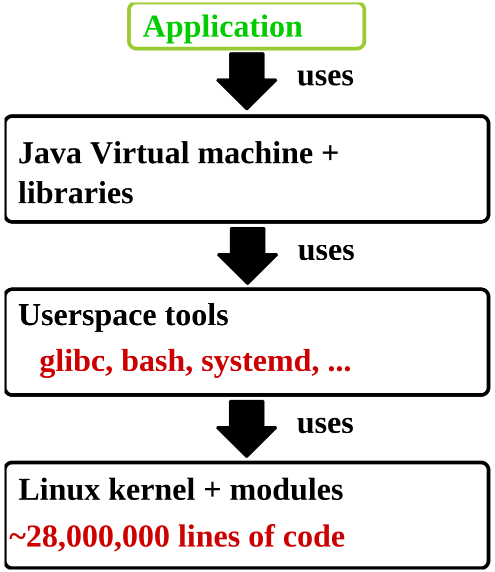
↓ **uses**

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

Application execution prerequisites



Application execution prerequisites



Tools:
-Compiler
-Debugger
-IDE
...

Application execution prerequisites

Application

uses

Java Virtual machine + libraries

uses

Userspace tools
glibc, bash, systemd, ...

uses

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

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

<table>
<tr><th>Source: <code>CreatePdf.java</code></th><th>PDF output</th></tr>
<tr><td><pre>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();</pre></td><td>Hello World!</td></tr>
</table>

Maven iText library pom.xml definition

IText Core » 8.0.3

A Free Java-PDF library

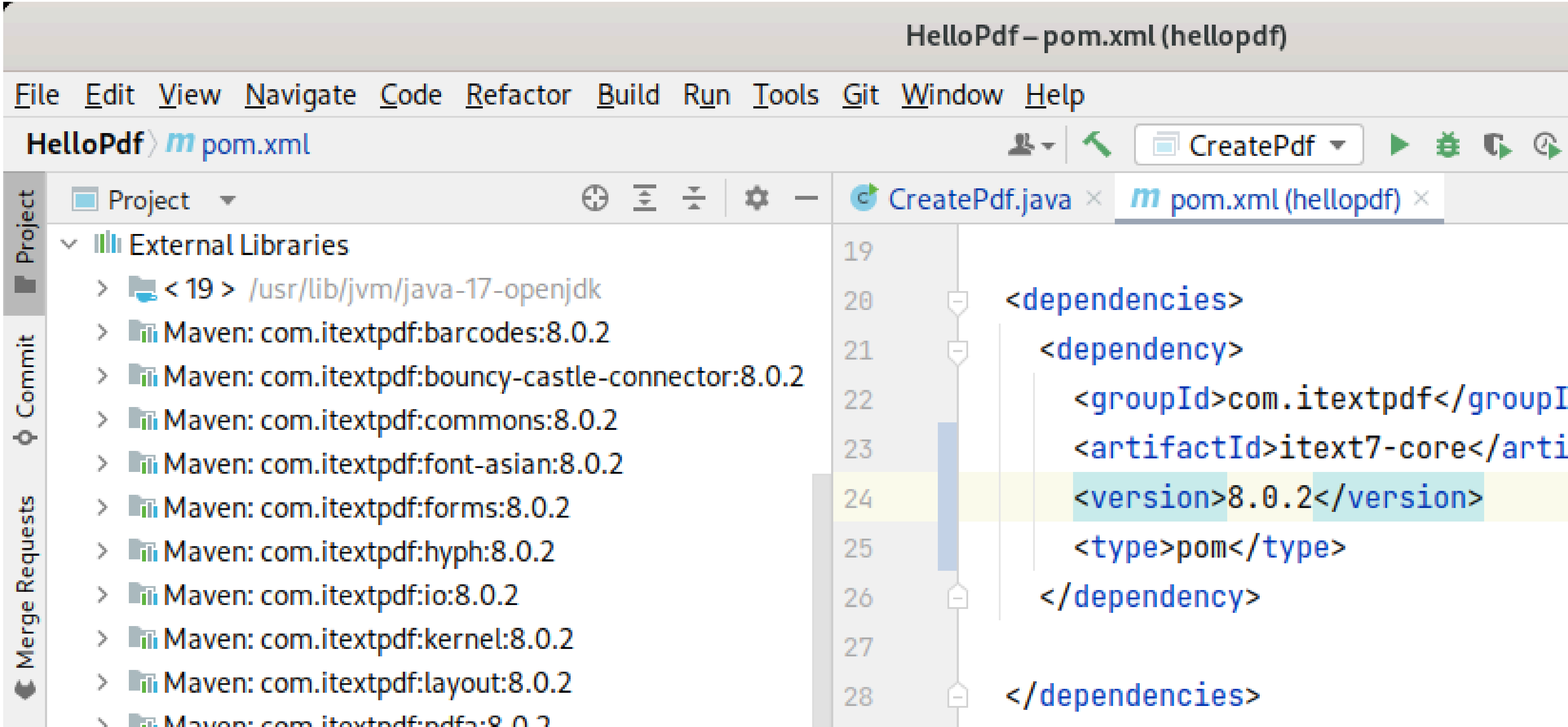
License	AGPL 3.0
Tags	pdf
Organization	Apryse Group NV
HomePage	https://itextpdf.com/
Date	Feb 07, 2024
Files	pom (4 KB) View All
Repositories	Central

```xml
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>com.itextpdf</groupId>
      <artifactId>itext-core</artifactId>
      <version>8.0.3</version>
      <type>pom</type>
    </dependency>
  </dependencies>
...

</project>
```

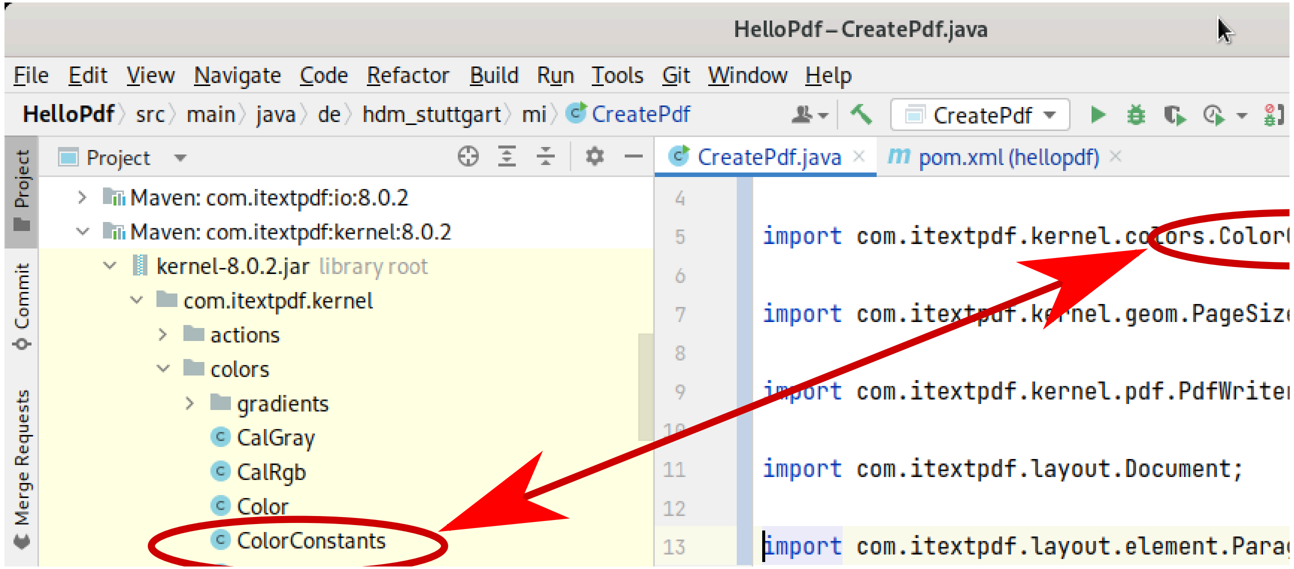
Itext transitive dependencies

```
mvn dependency:tree
...
 --- dependency:3.6.1:tree (default-cli) @ hellopdf ---
 de.hdm_stuttgart.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



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 -> am.ik.archetype:elm-spring-boot-blank-archety
2: remote -> am.ik.archetype:graalvm-blank-archetype ...
3: remote -> am.ik.archetype:graalvm-springmvc-blank-arche
...
2083: remote -> org.apache.maven.archetypes:maven-archetyp
...
3330: remote -> za.co.absa.hyperdrive:component-archetype_
Choose a number or apply filter ...
```

Project «lottery» depending on «helper»

Project »helper« ← Uses — 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

```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

```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
  /ma/goik/.m2/repository/mi/calc/common/1.0/common-1.0.jar
```

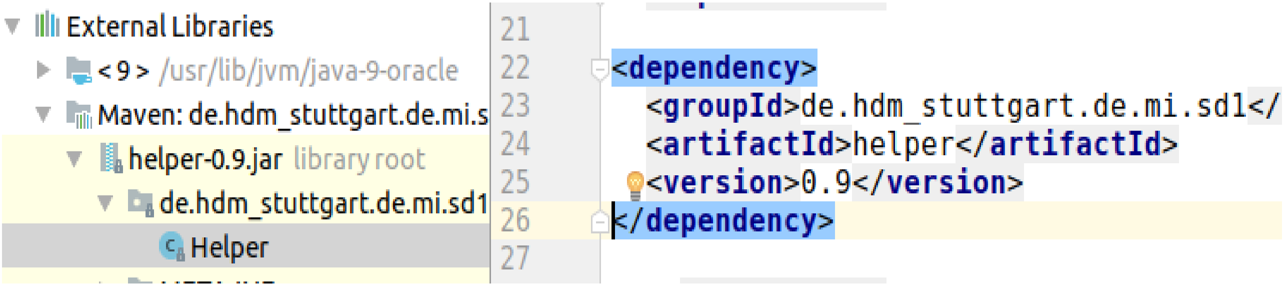
helper-0.9.jar archive content

```
goik@goiki tmp> unzip  ...hdm_stuttgart/de/mi/sd1/helper/0.9/helper-0.9.jar
Archive:  .../.m2/repository/.../sd1/helper/0.9/helper-0.9.jar
   creating: META-INF/
  inflating: META-INF/MANIFEST.MF
   creating: de/
   creating: de/hdm_stuttgart/
   creating: de/hdm_stuttgart/mi/
   creating: de/hdm_stuttgart/mi/sd1/
  inflating: de/hdm_stuttgart/mi/sd1/Helper.class
   creating: META-INF/maven/
   creating: META-INF/maven/de.hdm_stuttgart.mi.sd1/
   creating: META-INF/maven/de.hdm_stuttgart.mi.sd1/helper/
  inflating: META-INF/maven/de.hdm_stuttgart.mi.sd1/helper/pom.xml
  inflating: META-INF/maven/de.hdm_stuttgart.mi.sd1/helper/pom.properties
```

Consuming project «Lottery»

```
<project ...>
  ...
  <groupId>de.hdm_stuttgart.mi.sd1</groupId>
  <artifactId>lottery</artifactId>
  <version>0.9</version>

  <packaging>jar</packaging>
  <name>lottery</name>

  <dependencies>
    <dependency>
      <groupId>de.hdm_stuttgart.mi.sd1</groupId>
      <artifactId>helper</artifactId>
      <version>0.9</version>
    </dependency>
...
</project>
```

External libraries view



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;
  }}
```

mvn install

```
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;
  }}
```

mvn install

```
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>
```

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

Maven artifact dependency.

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

mvn install

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

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

mvn compile

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

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

Related exercises

Exercise 108: Cancelling fractions
Exercise 109: Dealing with local Maven dependencies

Using the MI Sd1 project template

- Download file `mavenTemplate.zip` from `here`.
- Extract `mavenTemplate.zip` to folder `template`.
- Optional: Edit `template/pom.xml` reflecting your project needs i.e. `<groupId>` and related.
- Optional: Import your project in Intellij IDEA.

CLI example

```
mvn --batch-mode -e archetype:generate \
-DgroupId=de.hdm_stuttgart.mi.sd1 -DartifactId=second -Dversion=0.9 \
-DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion

[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.hdm_stuttgart.mi \ ❸
 -DarchetypeArtifactId=mi-maven-archetype-quickstart \
 -DarchetypeVersion=2.3 \
                        \
 -DgroupId=de.hdm_stuttgart.mi.sd1 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 /ma/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: /ma/goik/first/target/first-0.9.jar
...
[INFO] Replacing /ma/goik/first/target/first-0.9.jar with
         /ma/goik/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 `javadoc:javadoc`

```
> mvn javadoc:javadoc
[INFO] Scanning for projects...
...
Generating /ma/goik/First/target/site/apidocs/allclasses-noframe.html...
Generating /ma/goik/First/target/site/apidocs/index.html...
Generating /ma/goik/First/target/site/apidocs/overview-summary.html...
Generating /ma/goik/First/target/site/apidocs/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 /ma/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`.

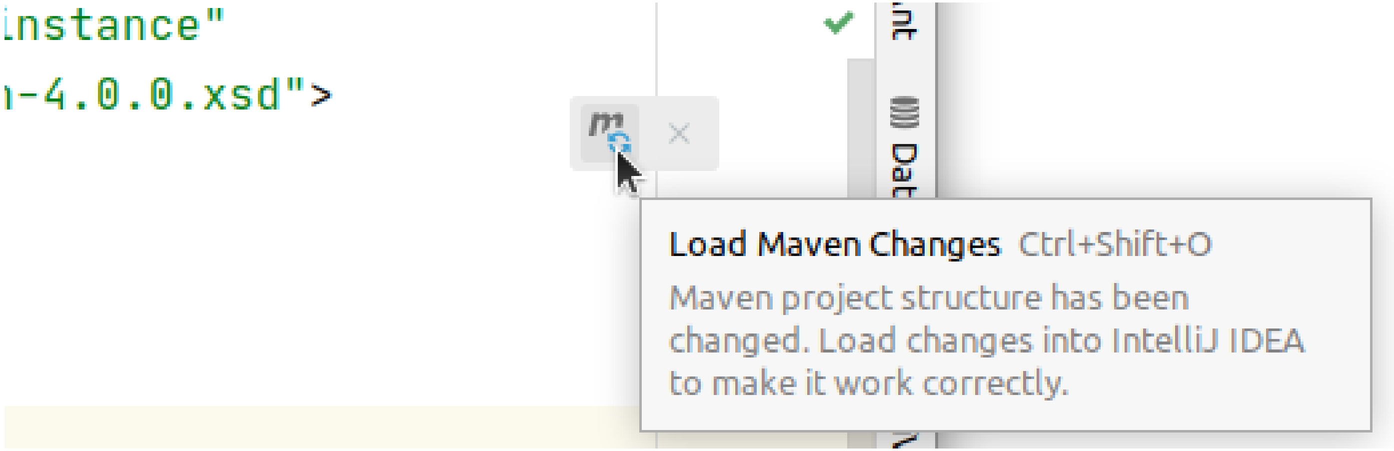
Adding MI Maven server

Adding MI Maven server

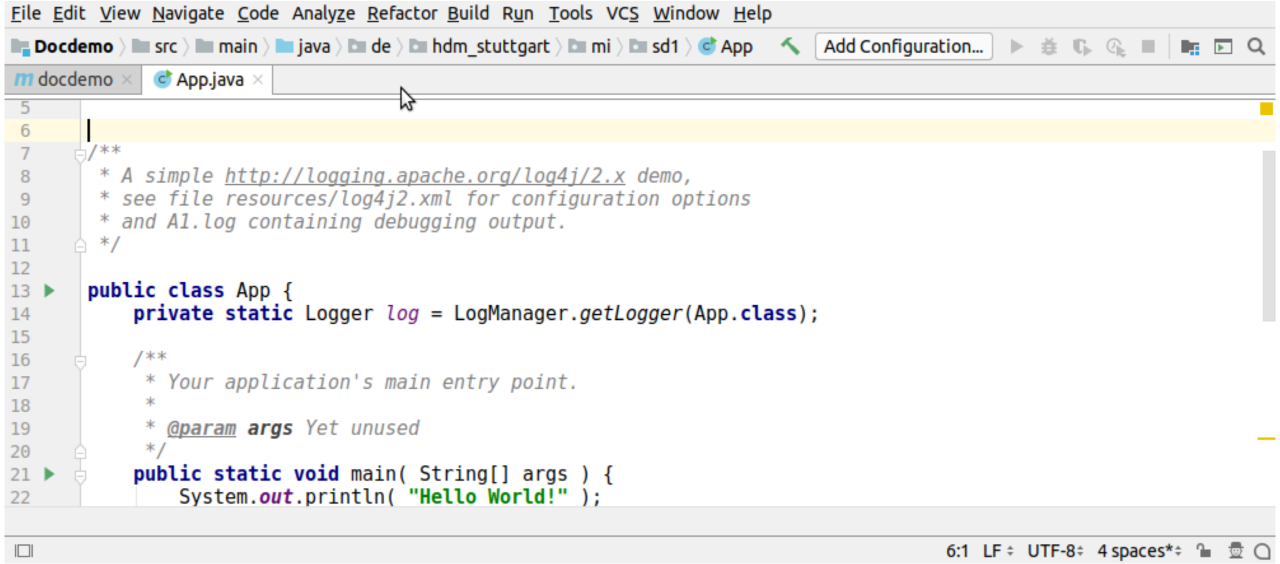
Adding MI Maven server

New MI archetype project

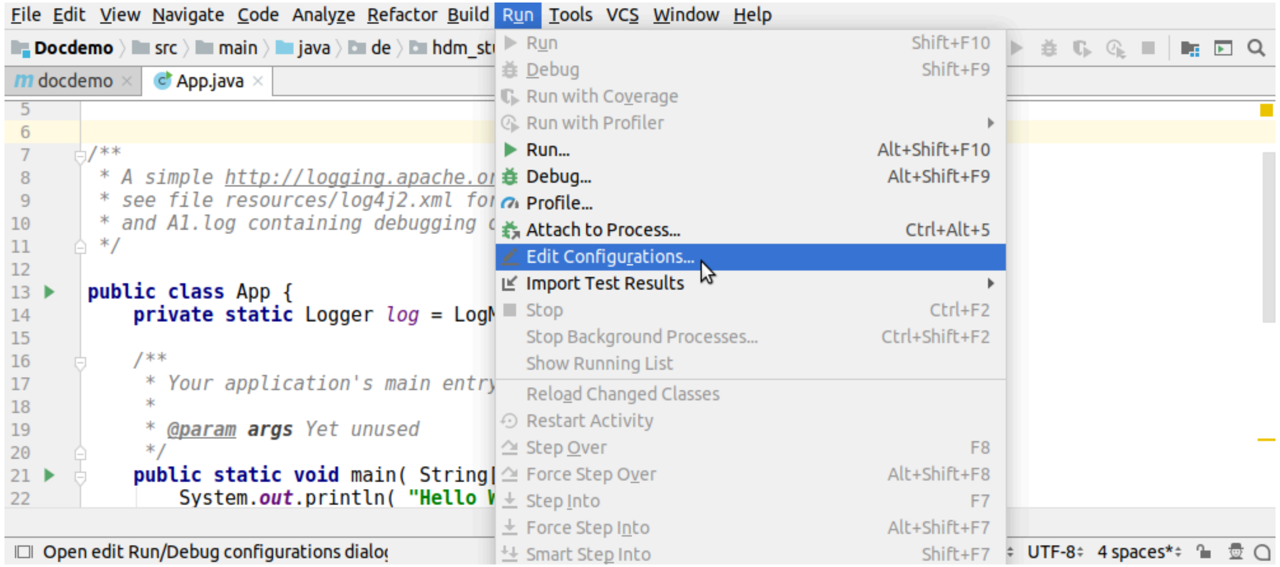
`pom.xml` content changes



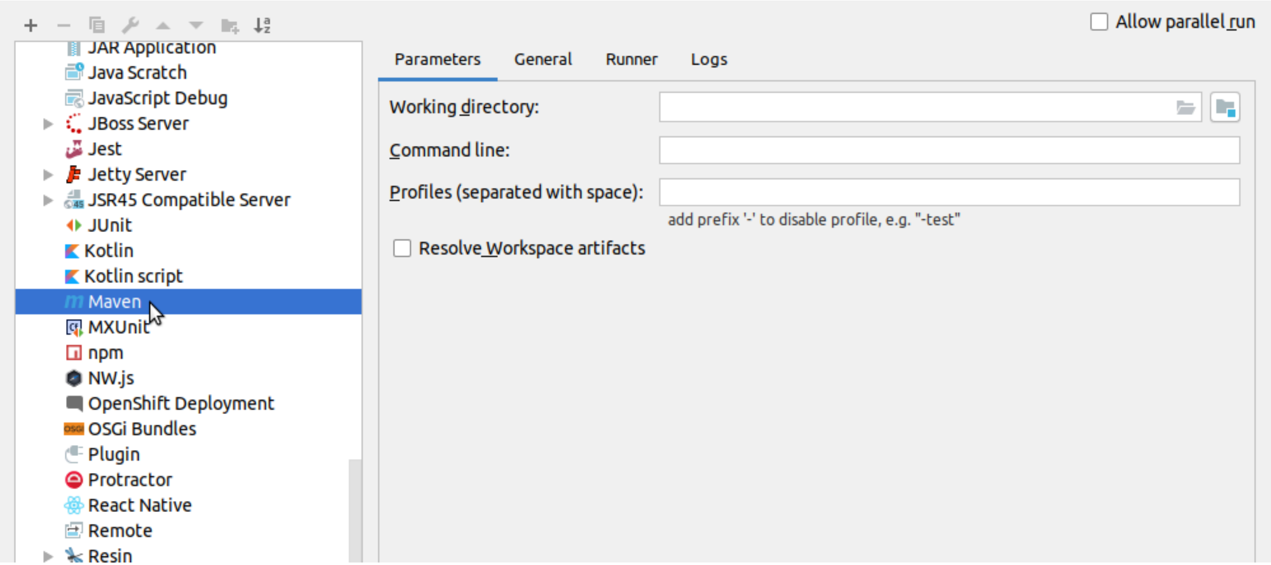
Intellij IDEA generating Javadoc™



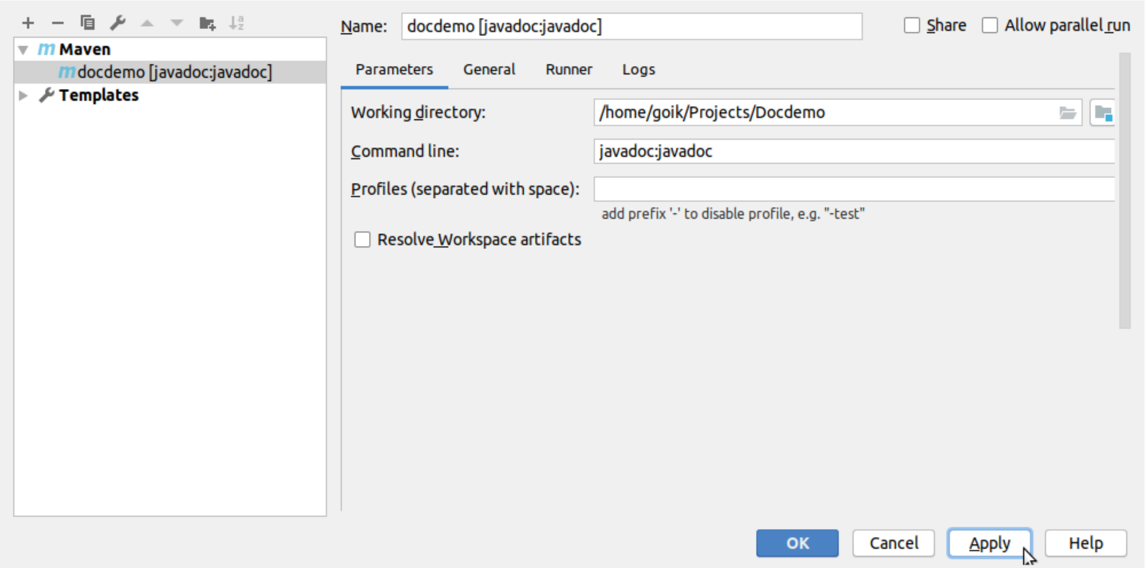
Intellij IDEA generating Javadoc™



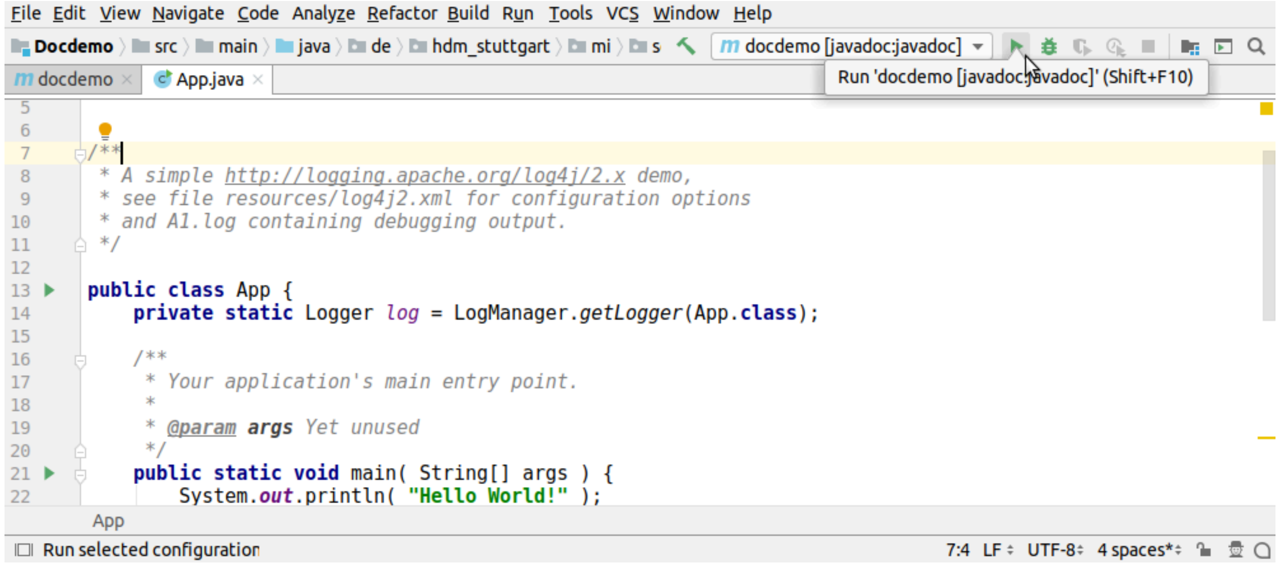
Intellij IDEA generating Javadoc™



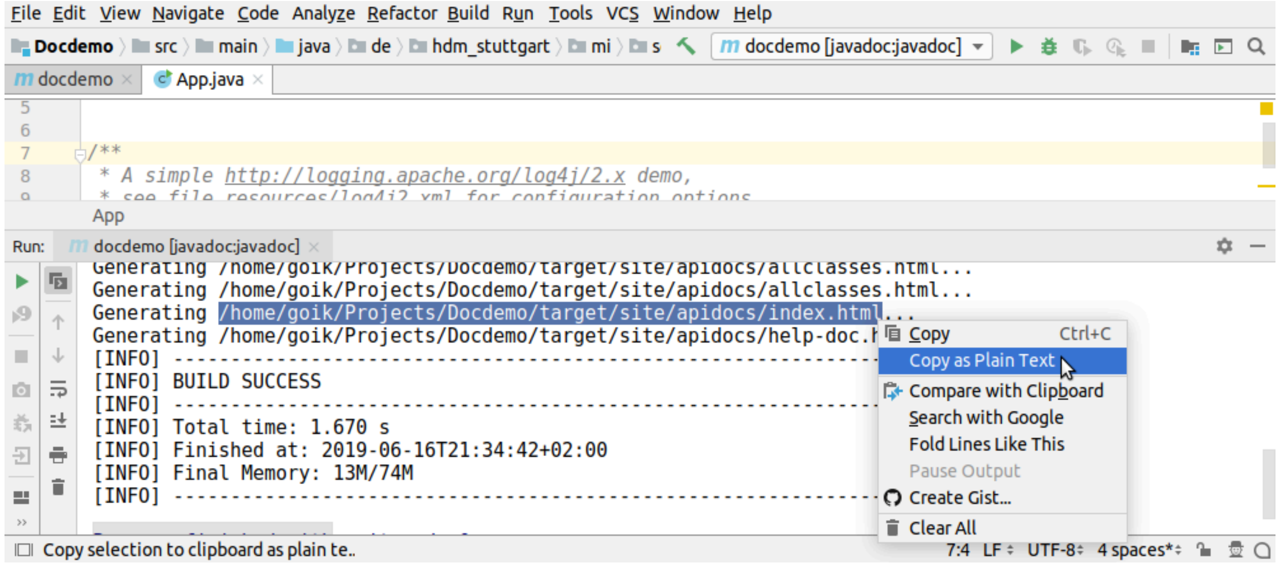
Intellij IDEA generating Javadoc™



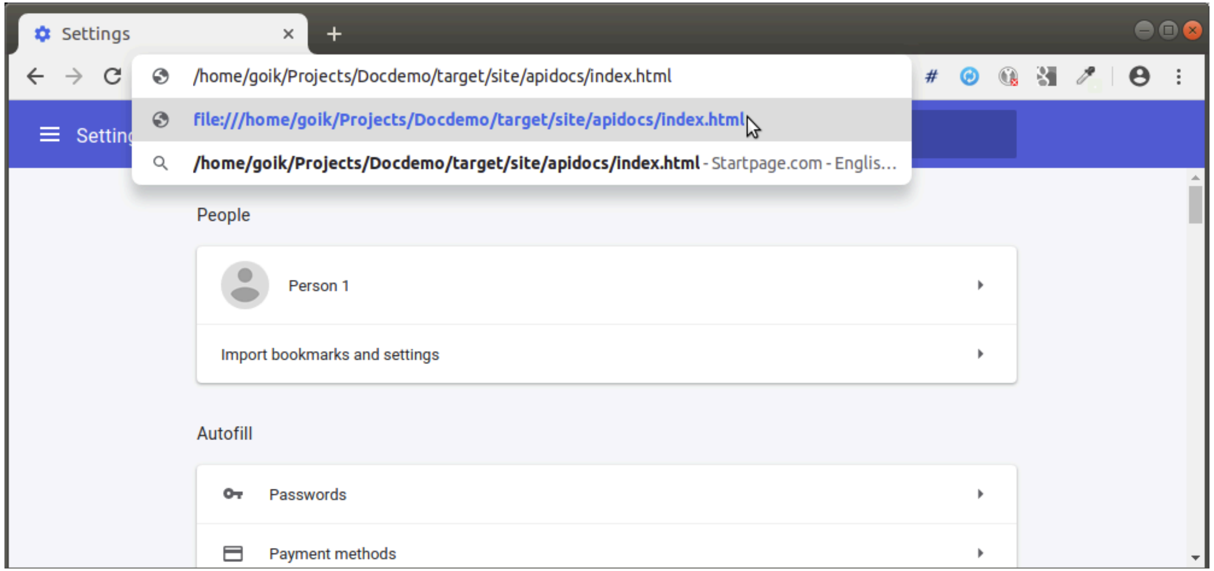
Intellij IDEA generating Javadoc™



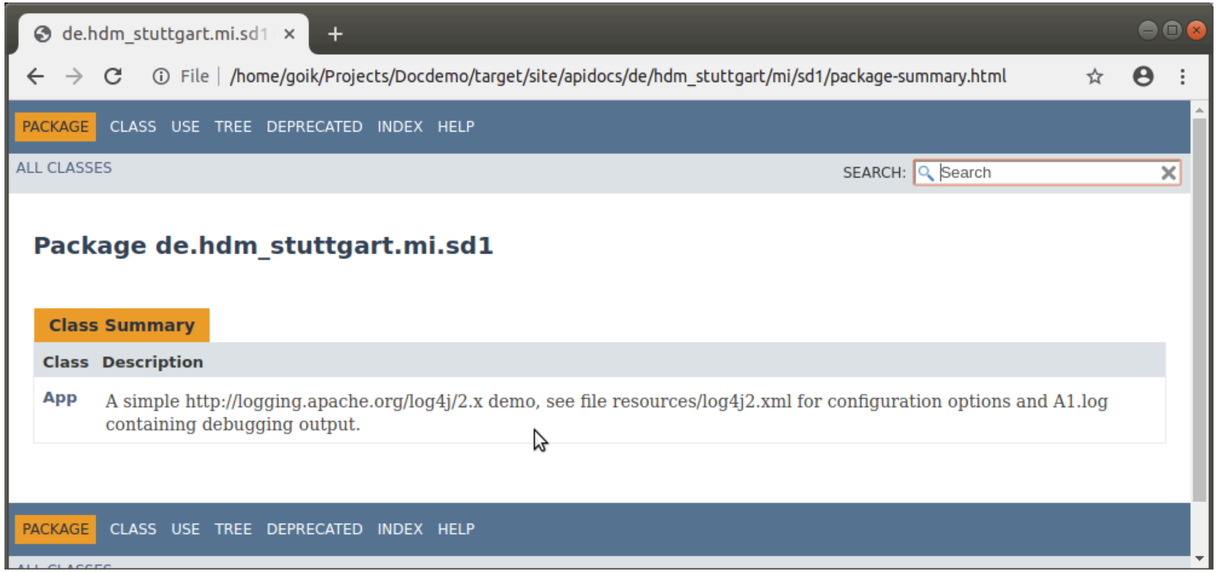
Intellij IDEA generating Javadoc™



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

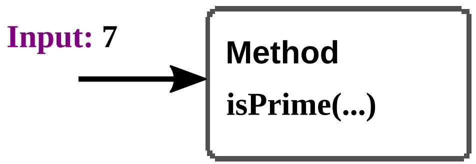
Testing method isPrime(int n)

Method

isPrime(...)

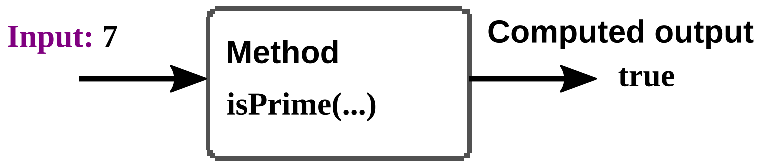
Unit test principle

Testing method isPrime(int n)



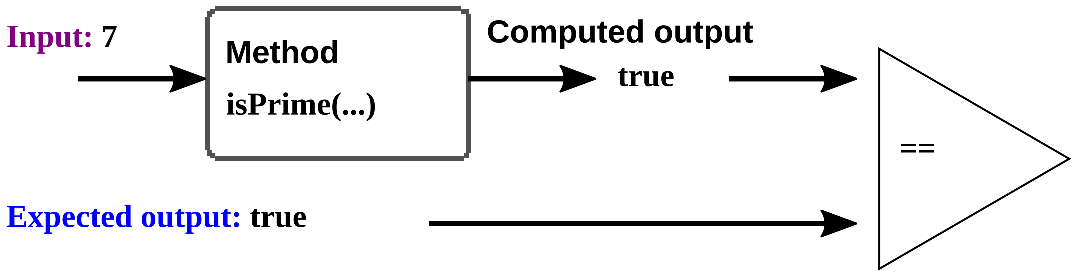
Unit test principle

Testing method isPrime(int n)



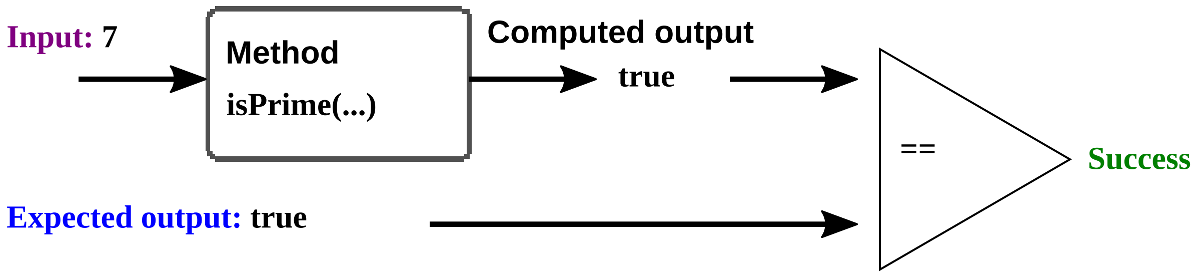
Unit test principle

Testing method isPrime(int n)



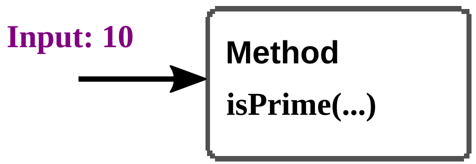
Unit test principle

Testing method isPrime(int n)



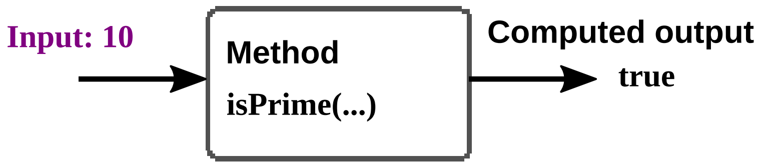
Unit test principle

Testing method isPrime(int n)



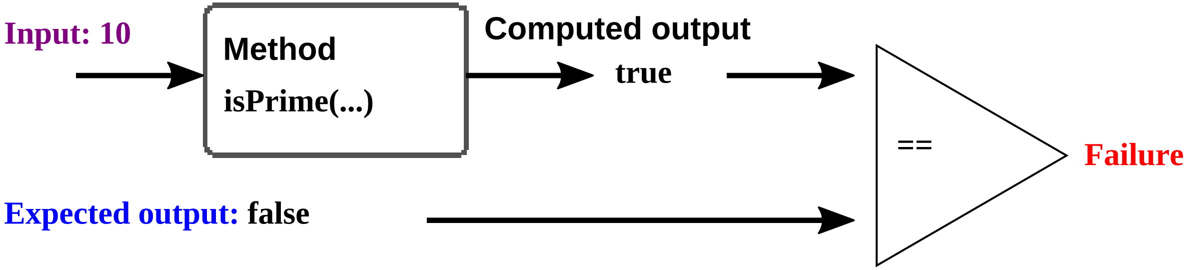
Unit test principle

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`

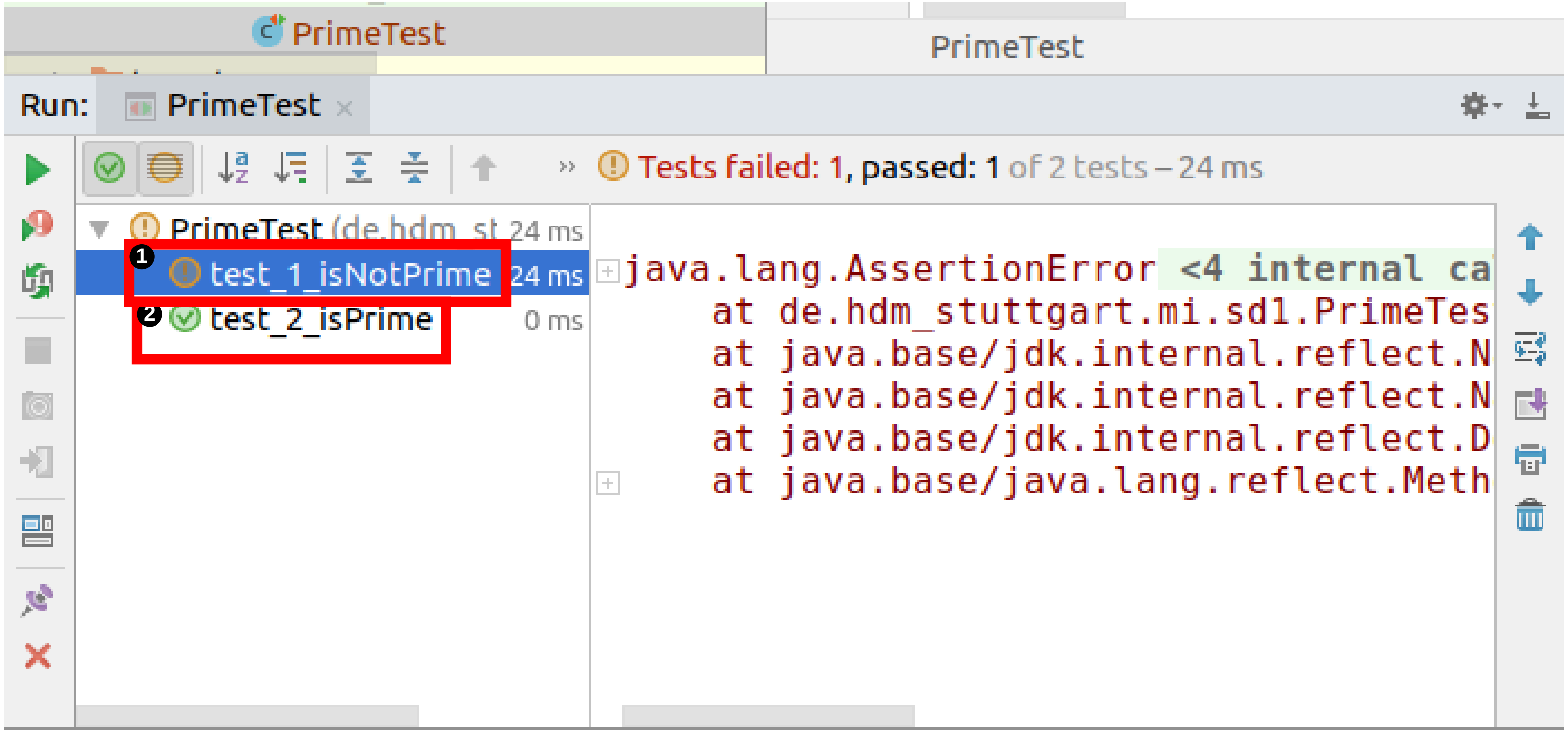
Step 3: Junit based specification test

```
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 test_Primes() {
  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

```java
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.hdm_stuttgart.mi.sd1.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

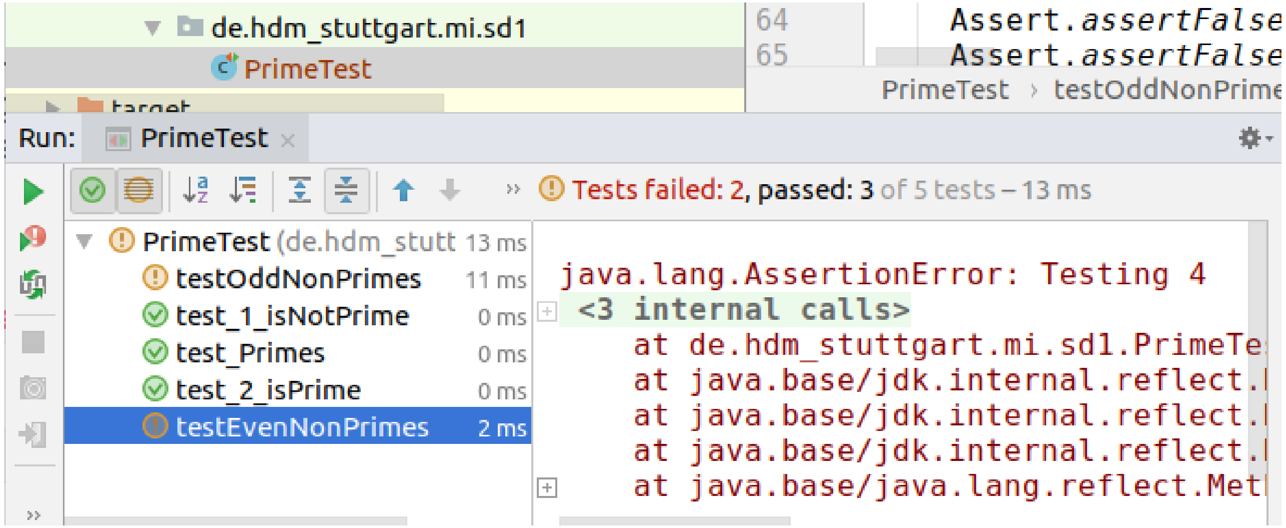
```
101 / 2  = 50 remainder 1
101 / 3  = 33 remainder 2
101 / 4  = 25 remainder 1
101 / 5  = 20 remainder 1
101 / 6  = 16 remainder 5
101 / 7  = 14 remainder 3
101 / 8  = 12 remainder 5
101 / 9  = 11 remainder 2
101 / 10  = 10 remainder 1
101 / 11  = 9 remainder 2
101 / 12  = 8 remainder 5
101 / 13 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

Changing the implementation

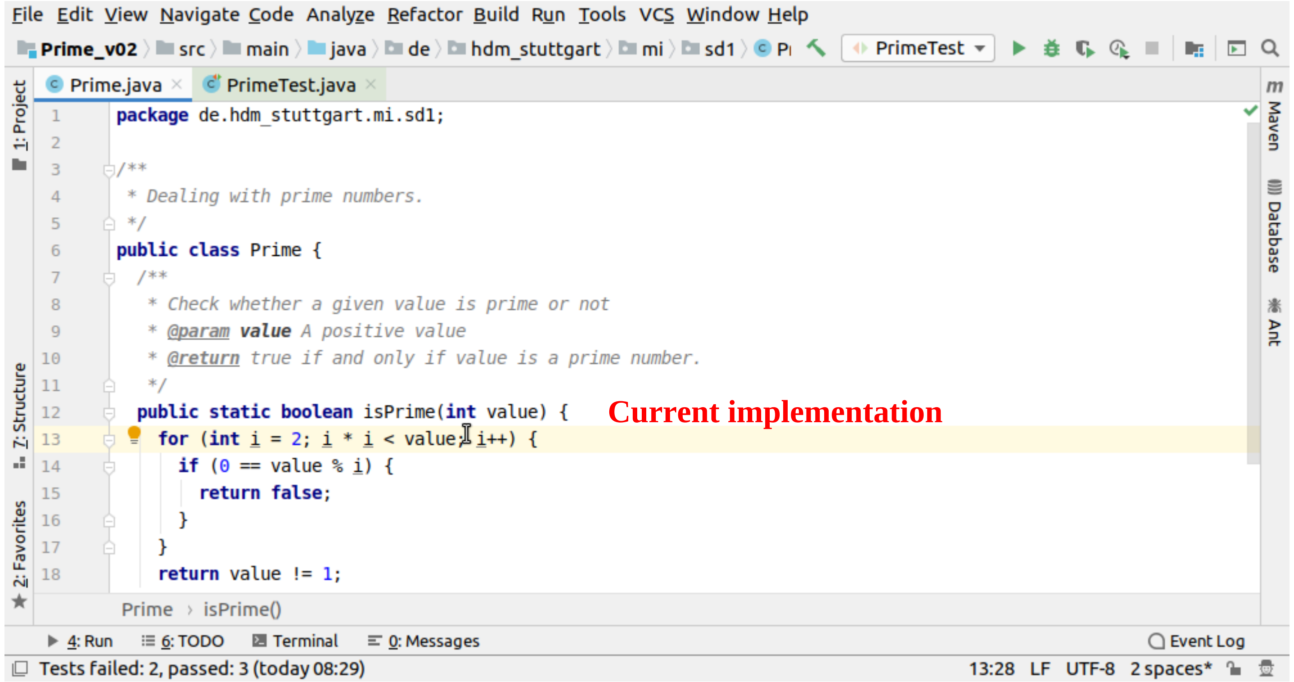
Big performance gain:

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

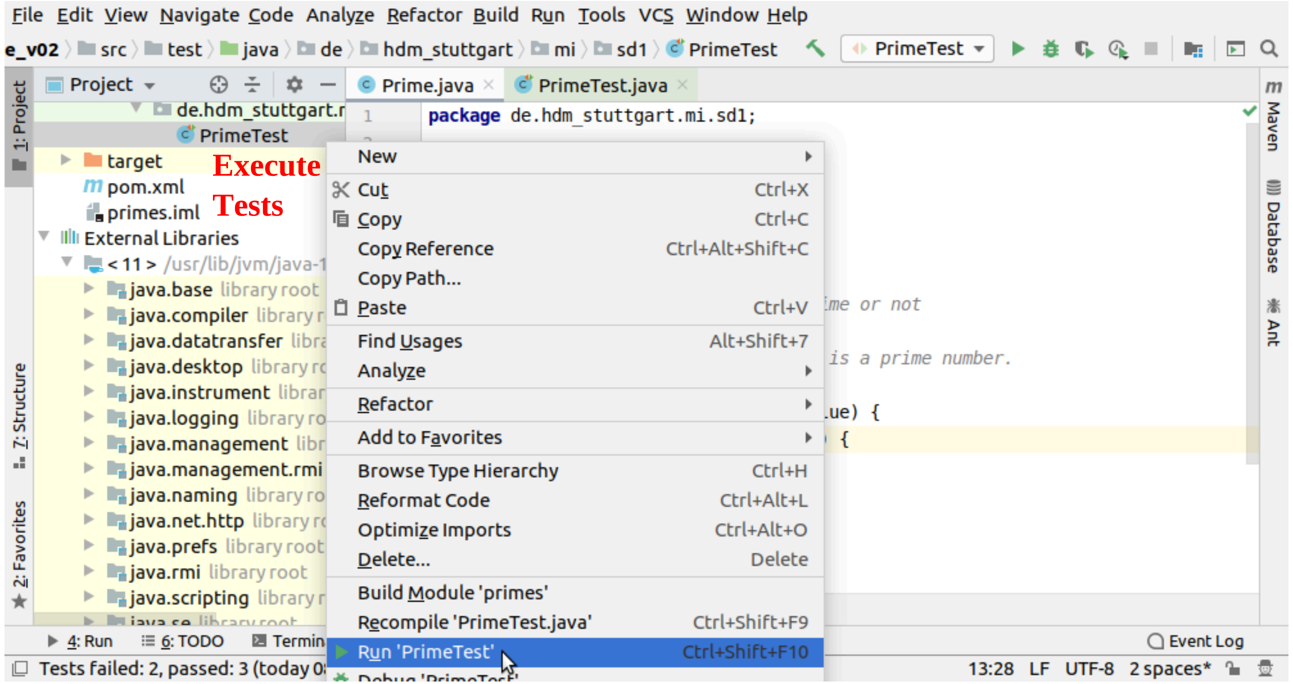
Regression test



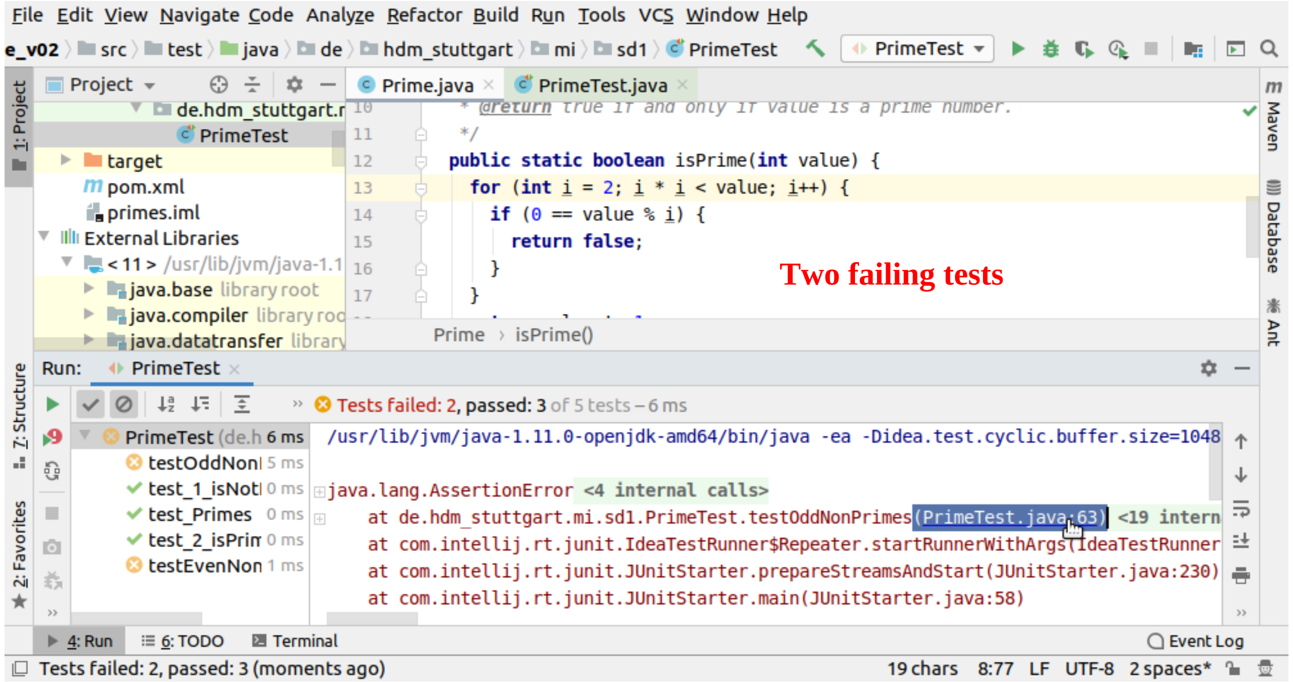
Systematic error debugging



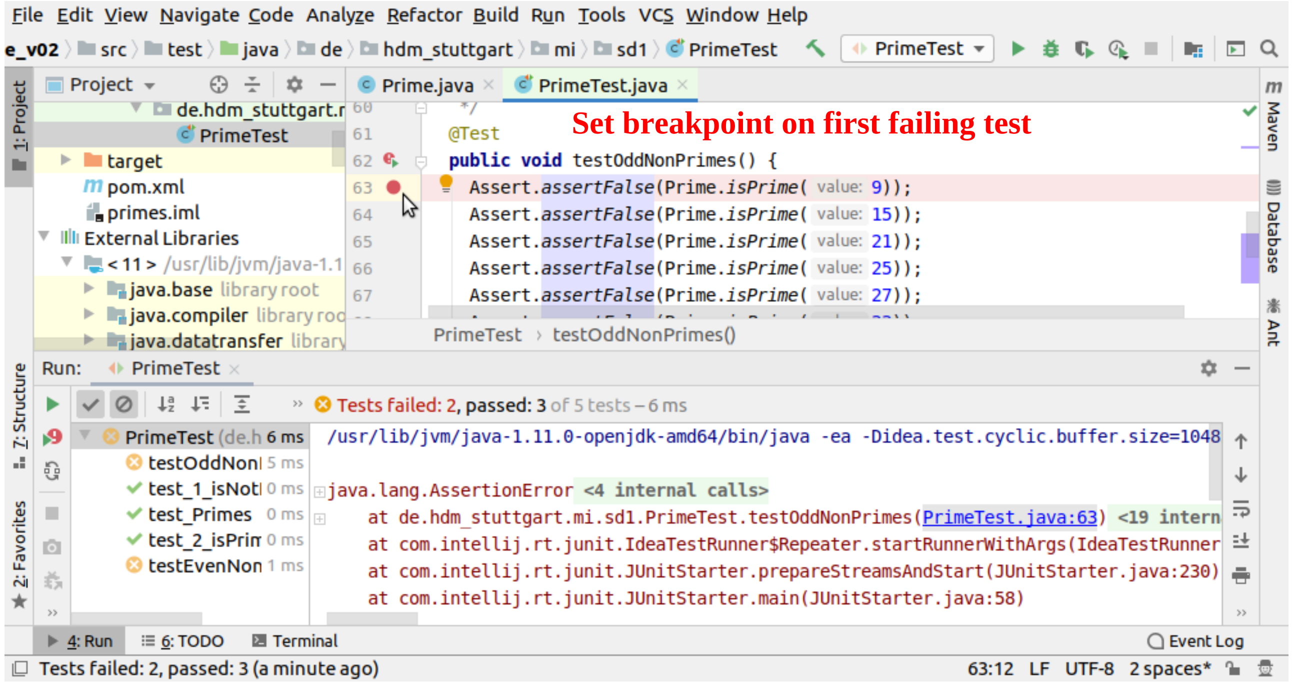
Systematic error debugging



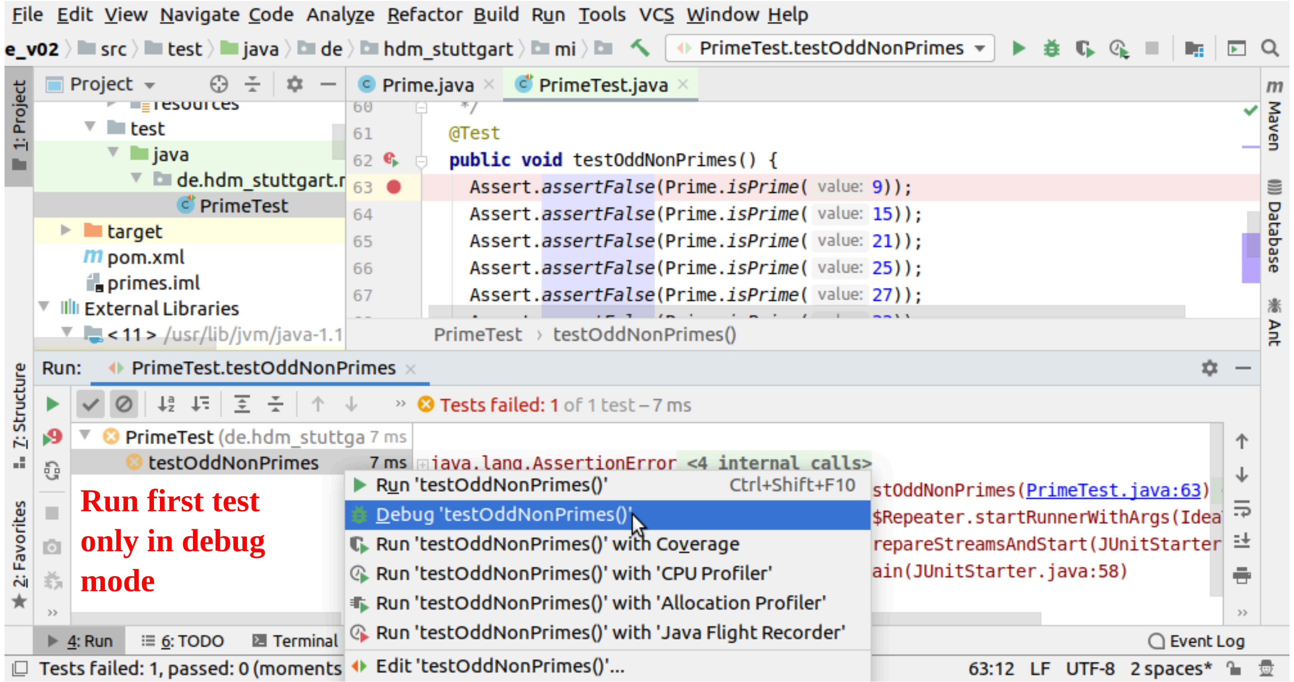
Systematic error debugging



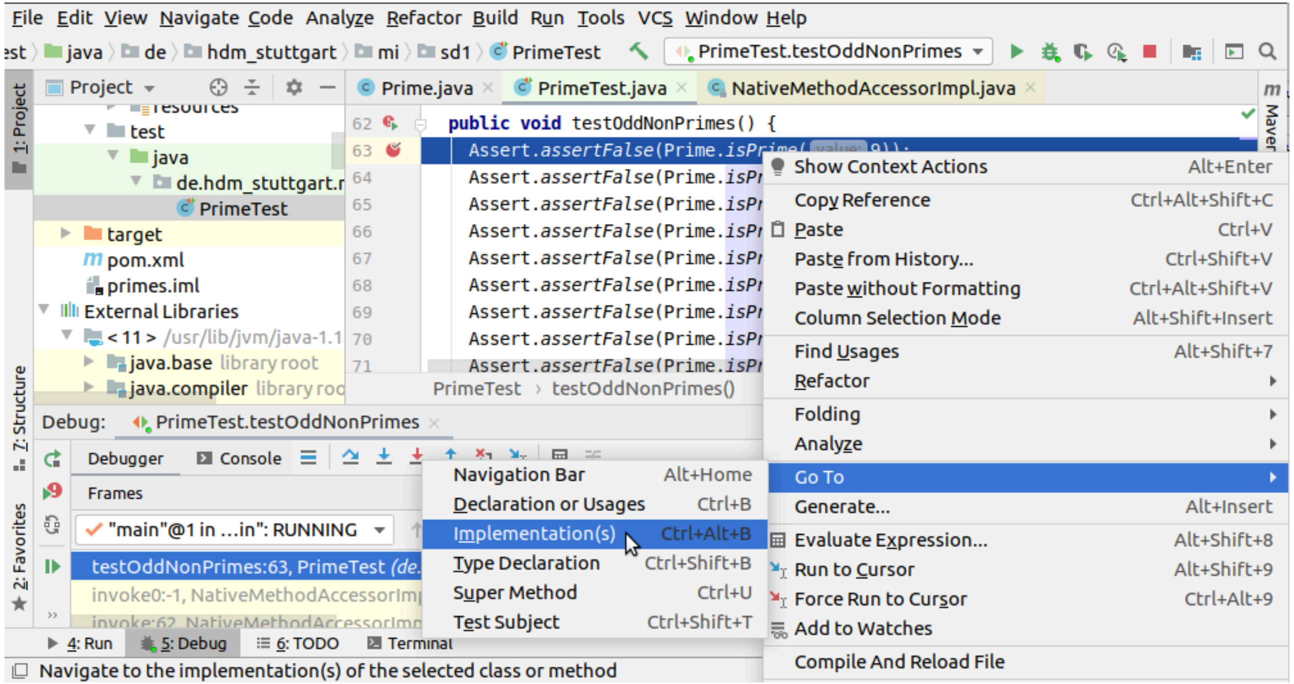
Systematic error debugging



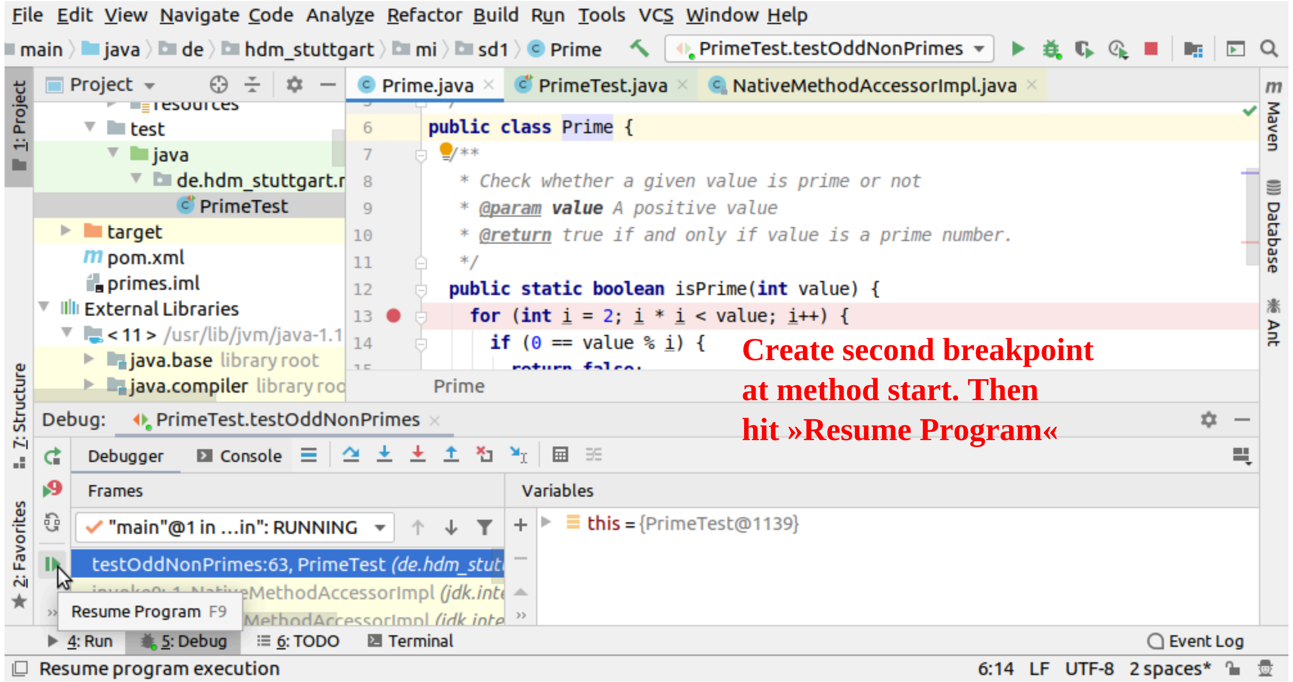
Systematic error debugging



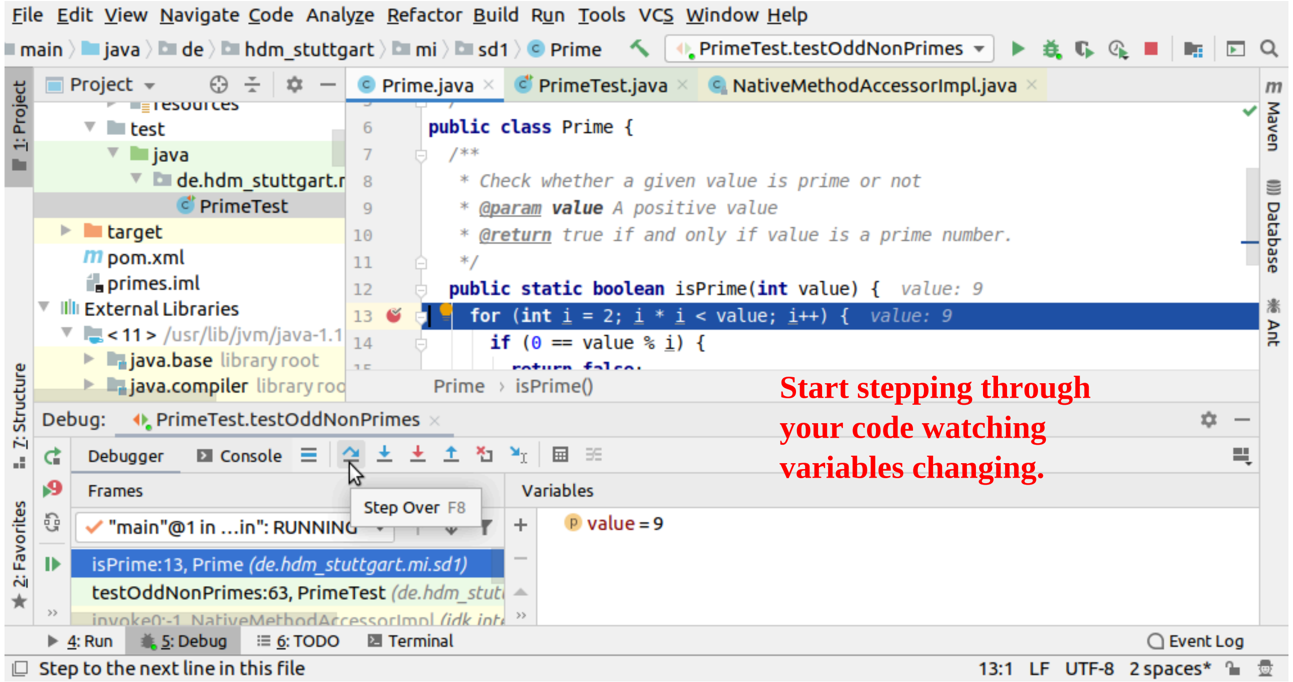
Systematic error debugging



Systematic error debugging

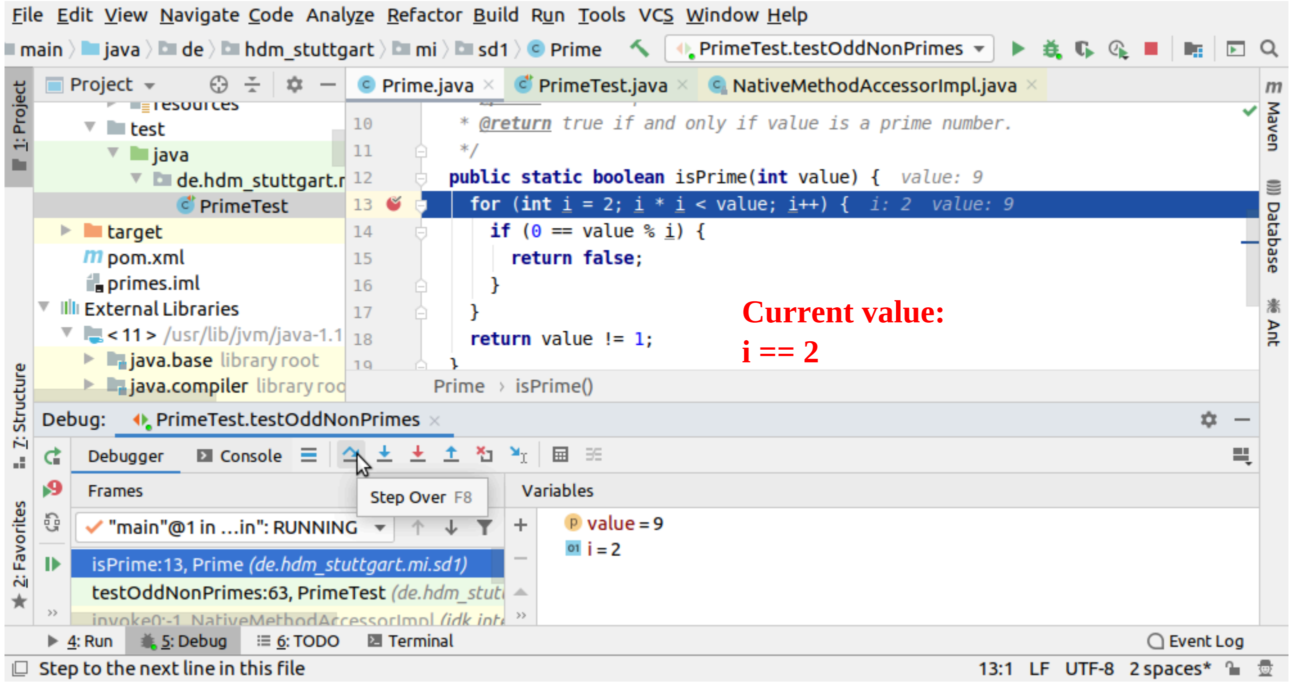


Systematic error debugging

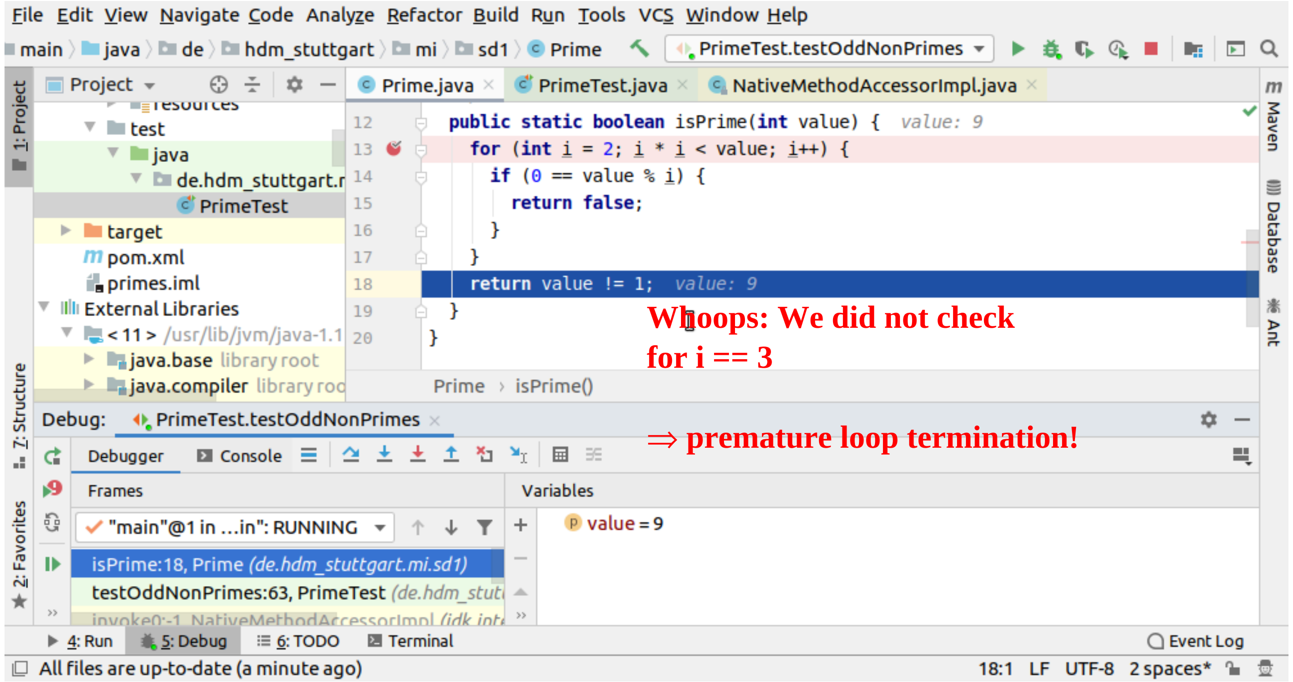


Start stepping through your code watching variables changing.

Systematic error debugging



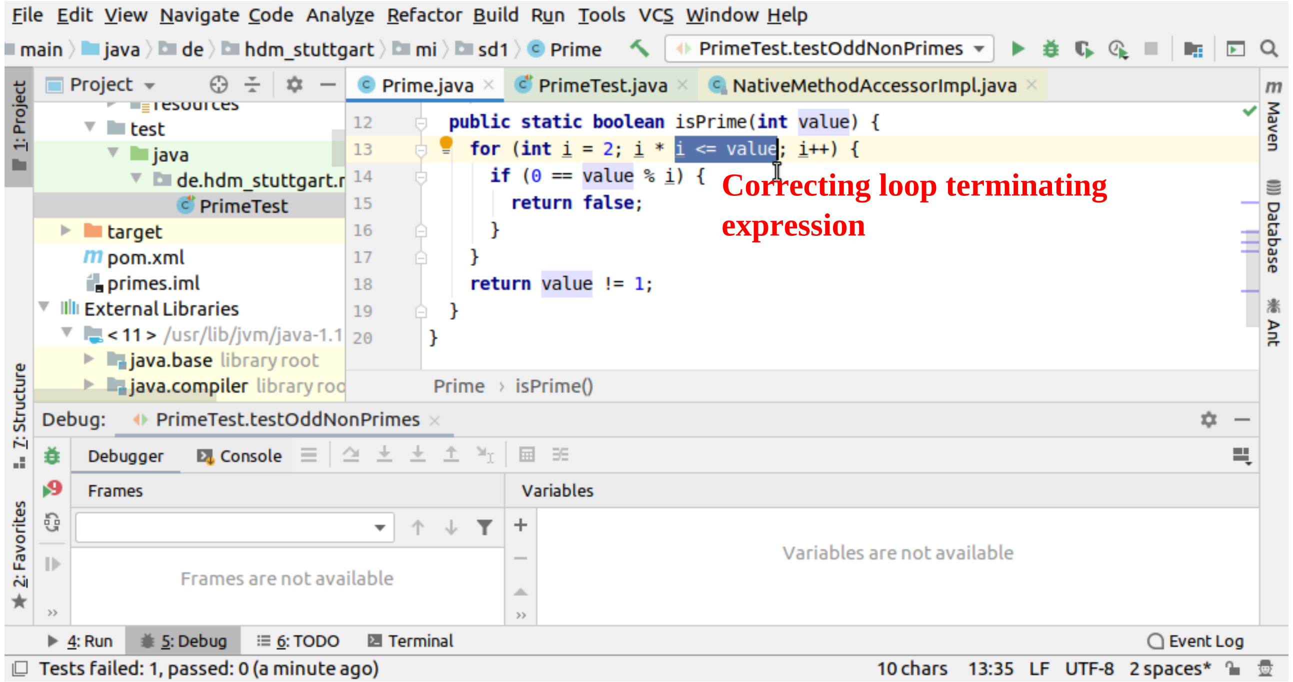
Systematic error debugging



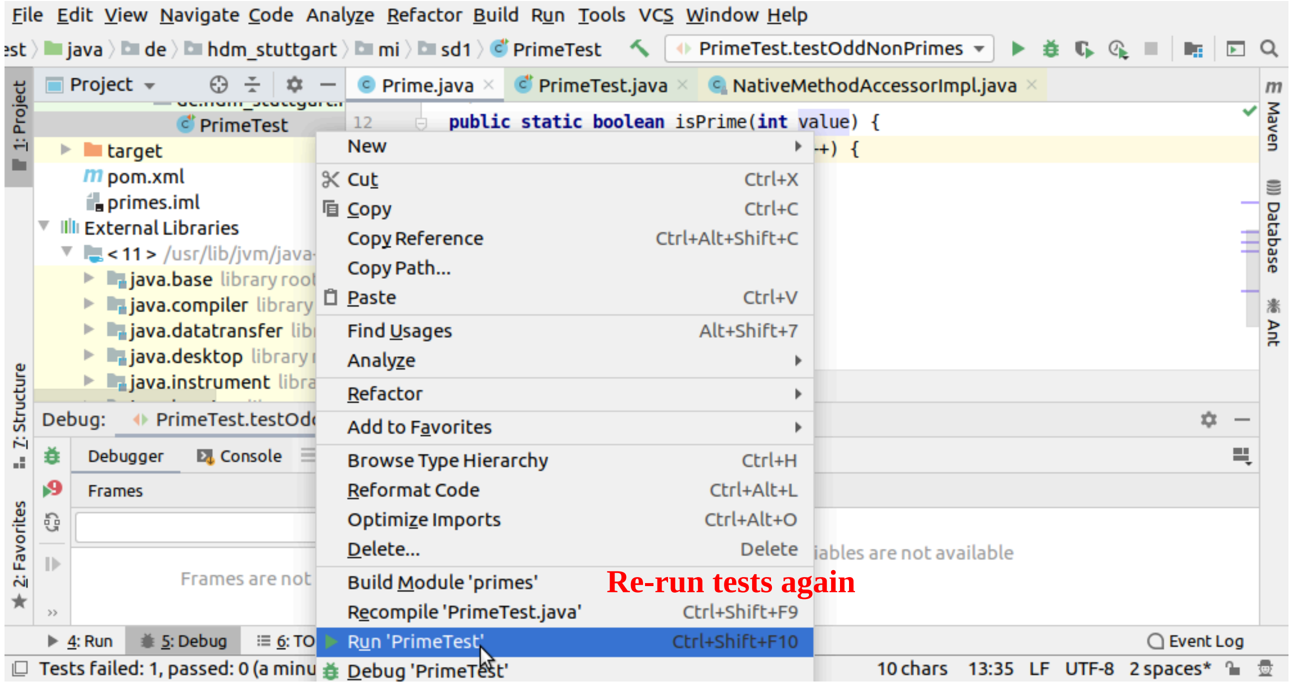
Whoops: We did not check for i == 3

⇒ premature loop termination!

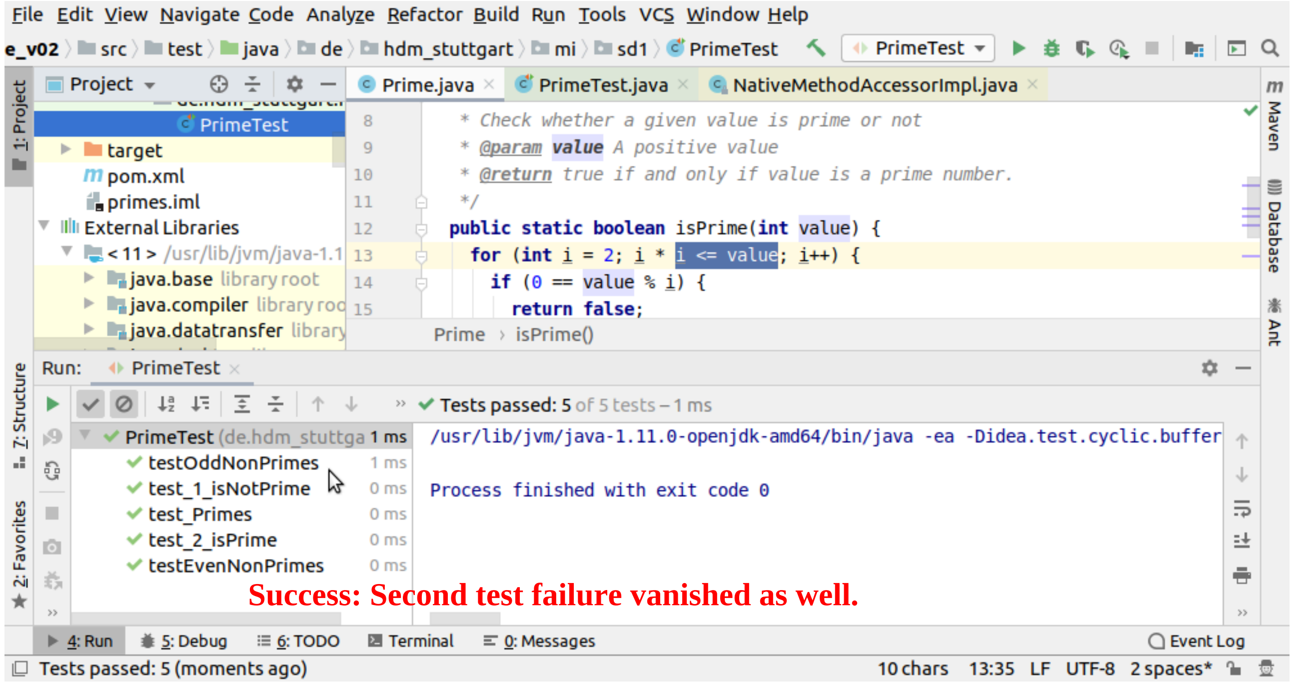
Systematic error debugging



Systematic error debugging



Systematic error debugging



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` !!

```
/**
 * Comparing double values
 */
public class doubleCompareTest {
  /**
   * Comparing 3.6 and 3.6 ?
   */
  @Test
  public void test1_3() {
    Assert.assertEquals( expected: 3.6,  actual: 3 * 1.2 );
  }
```

'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	`int a = 1;`
	`int b = a;`
	`a = 5;`
	`System.out.println("a=" + a);`
	`System.out.println("b=" + b);`
		a=5
		b=1

Reference type	`StringBuffer r = new StringBuffer("Joe");`
	`StringBuffer s = r;`
	`r.append(" Simpson");`
	`System.out.println("r=" + r);`
	`System.out.println("s=" + s);`
		r=Joe Simpson
		s=Joe Simpson

Value variable Details

Stack

Heap

No objects
involved

```
int a = 1;

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

```
int a = 1;

int b = a;

a = 5;

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

a: 1

No objects
involved

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

Value variable Details

Stack

Heap

```
int a = 1;

int b = a;

a = 5;

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

a: 1

copy

b: 1

No objects involved

Assign new value 5 to variable a.
Independent variable b does not change.

Value variable Details

Stack

a: 5

b: 1

Heap

No objects involved

```
int a = 1;

int b = a;

a = 5;

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

Print result showing:

a=5
b=1

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

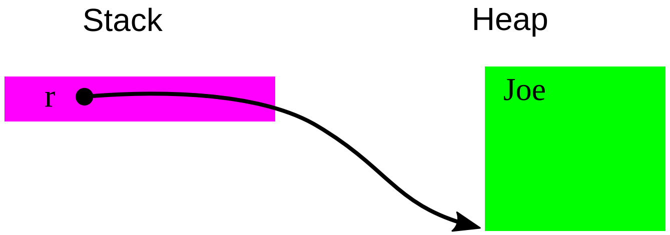
Joe

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

StringBuffer s = r;

r.append(" Simpson");

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

Assign heap reference to stack variable r.

Reference variable Details

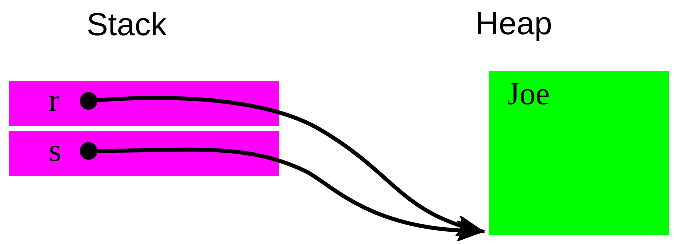
```
StringBuffer r =
  new StringBuffer("Joe");

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.

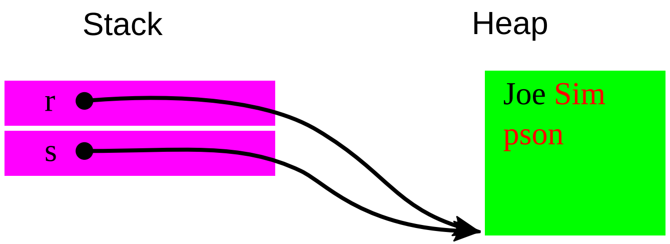
Reference variable Details

```
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™

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

```
Before printDuplicateValue: 3
printDuplicateValue: 6
After printDuplicateValue: 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);
}
```

Stack

main(...)	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);
}
```

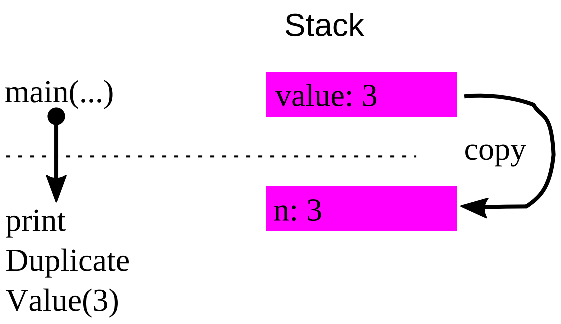
Stack

main(...)	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);
}
```



Duplicate value of 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);
}
```

Stack

main(...) value: 3

print Duplicate Value(3) n: 6

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);
}
```

Stack

main(...) 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

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

Stack

Heap

main(...)

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);}
```

Stack	Heap
main(...)	"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);}
```

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

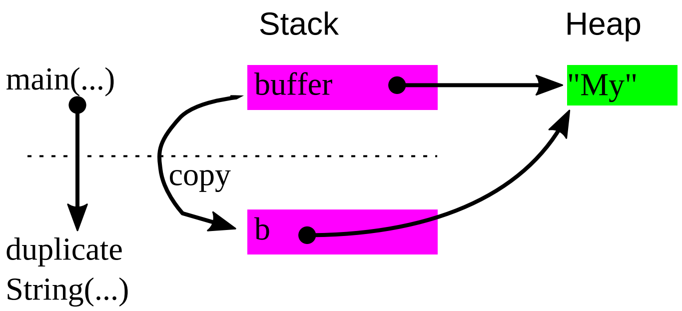
«call-by-reference» details

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

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);}
```

Using reference variable buffer for
printing StringBuffer heap instance.

No «call-by-reference» in Java™!

<pre>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");
}</pre>
	<pre>Before duplicateString: My
	After duplicateString: My</pre>

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");
}
```

Stack

Heap

main(...)

Create StringBuffer instance on heap.

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");
}
```

Stack

Heap

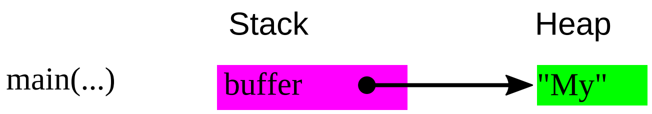
main(...)

"My"

Create reference variable buffer initializing with StringBuffer heap reference.

No «call-by-reference» details

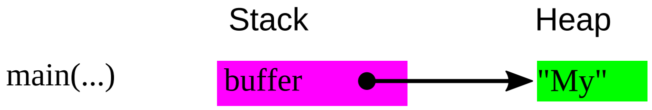
```
... main(...) {
  StringBuffer buffer
   =new StringBuffer("My");
  ...println(
   "Before: " + buffer);
  replaceString(buffer);
  ...println(
   "After: " + buffer);
}

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



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

No «call-by-reference» details

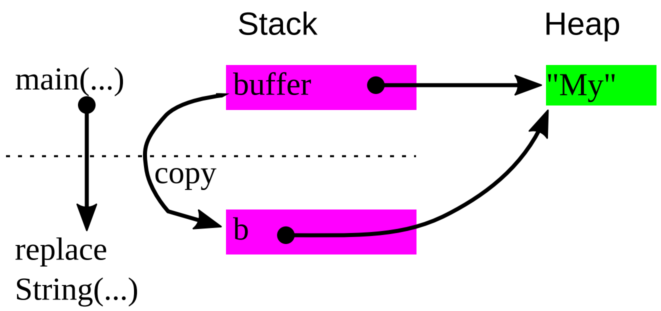
```
... main(...) {
  StringBuffer buffer
   =new StringBuffer("My");
  ...println(
   "Before: " + buffer);
  replaceString(buffer);
  ...println(
   "After: " + buffer);
}

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



Calling replaceString() method.

No «call-by-reference» details

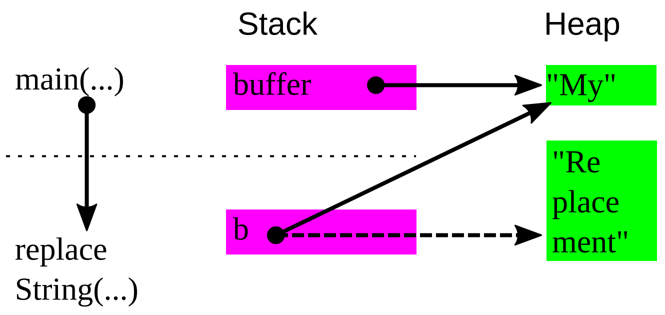
```
... main(...) {
  StringBuffer buffer
   =new StringBuffer("My");
  ...println(
   "Before: " + buffer);
  replaceString(buffer);
  ...println(
   "After: " + buffer);
}

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



Creating second instance on heap.

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");
}
```

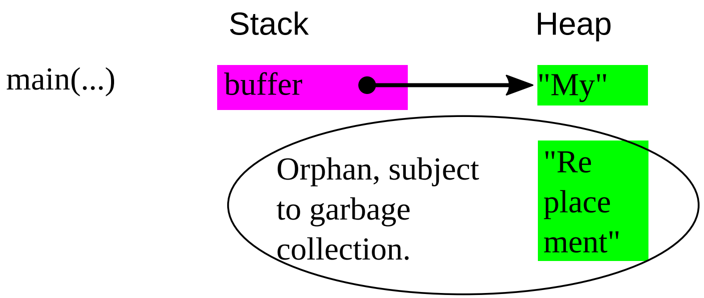


Changing reference to new heap instance.

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 “&”

<pre>// 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;
}</pre>
	<pre>Before call: 3
	duplicateValue: 6
	After call: 6</pre>

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 ...;
}
```

Stack

main(...)	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 ...;
}
```

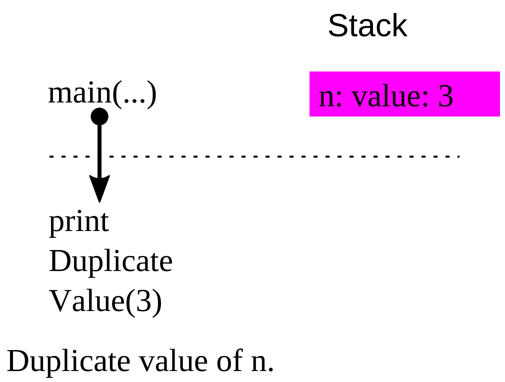
Stack

main(...) value: 3

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

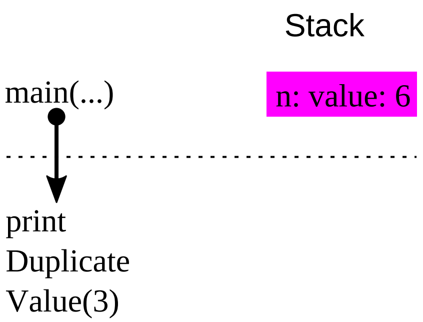
C++ «call-by-reference» details

```
... 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 ...;
}
```

Stack

main(...) 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 ➊) {
    double val ➋ = area  / PI ➌;
    return ➍ r ➎ = Math.sqrt(val);
  }
}
```

➊ Passing arguments.

➋ Defining method local variables.

➌ Accessing class variable.

➍ returning values.

➎ 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 (`static`)	Application process
Instance scope	Object lifetime: `new (...)` until being garbage collected.
Method scope	Method invocation until `return`.

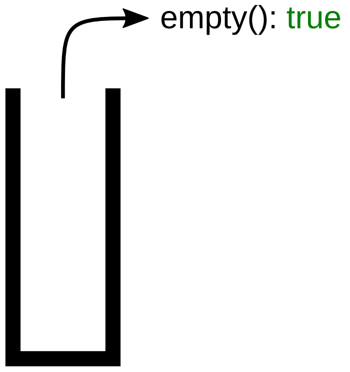
Two runtime memory categories

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

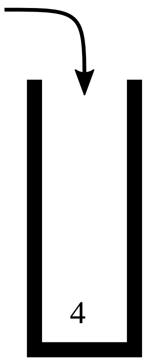
Method	Description	Precondition	Access type
`push(...)`	Add object on top	-	Modify
`pop()`	Read + remove topmost object	Stack not empty	Modify
`top()`	Read topmost object	Stack not empty	Read only
`empty()`	`true` if and only if stack is empty	-	Read only

Example: Storing integer values

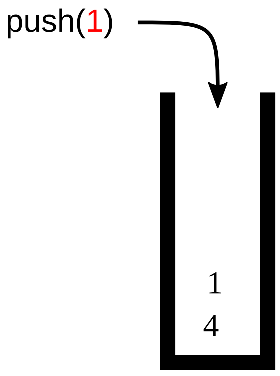


Example: Storing integer values

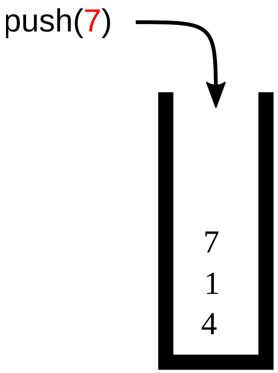
push(4)



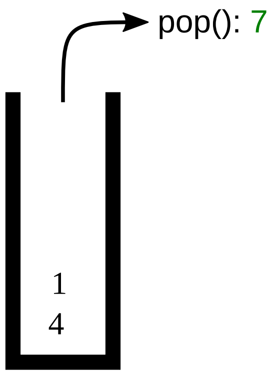
Example: Storing integer values



Example: Storing integer values

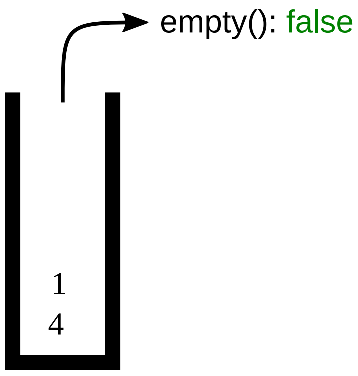


Example: Storing integer values



Example: Storing integer values

Example: Storing integer values



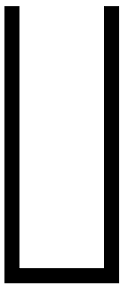
Method calling

```
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

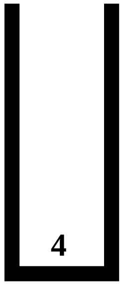
```
final Stack<Integer> si =
  new Stack<>();
si.push(4);
si.push(1);
si.push(7);
System.out.println("Top: " + si.peek()); // top
while (! si.empty()) {
  System.out.println("Not empty: " + si.pop());
}
```



Add integer value 4 to the stack

Method calling

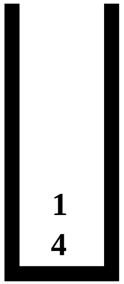
```
final Stack<Integer> si =
  new Stack<>();
si.push(4);
si.push(1);
si.push(7);
System.out.println("Top: " + si.peek()); // top
while (! si.empty()) {
  System.out.println("Not empty: " + si.pop());
}
```



Add integer value 1 to the stack

Method calling

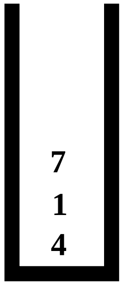
```
final Stack<Integer> si =
  new Stack<>();
si.push(4);
si.push(1);
si.push(7);
System.out.println("Top: " + si.peek()); // top
while (! si.empty()) {
  System.out.println("Not empty: " + si.pop());
}
```



Add integer value 7 to the stack

Method calling

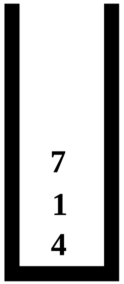
```
final Stack<Integer> si =
  new Stack<>();
si.push(4);
si.push(1);
si.push(7);
System.out.println("Top: " + si.peek()); // top
while (! si.empty()) {
  System.out.println("Not empty: " + si.pop());
}
```



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());
}
```

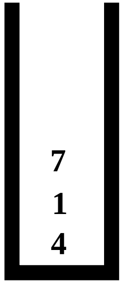


Check if stack is empty (not yet)

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());
}
```

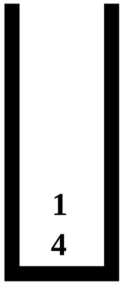


Remove 7 from stack and write its value to standard output.

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());
}
```

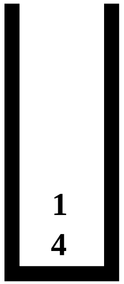


Check if stack is empty
(not yet)

```
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());
}
```

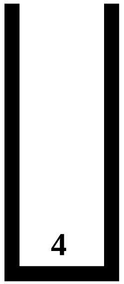


Remove **1** **from stack and write its value to standard output.**

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());
}
```

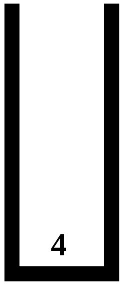


Check if stack is empty (not yet)

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());
}
```

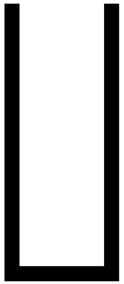


Remove 4 from stack and write its value to standard output.

```
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());
}
```

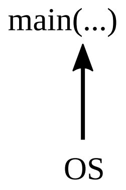


Check if stack is empty: true
(Program termination)

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);
  }
}
```

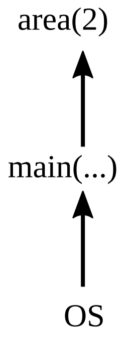
main(...)

OS

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);
  }
}
```



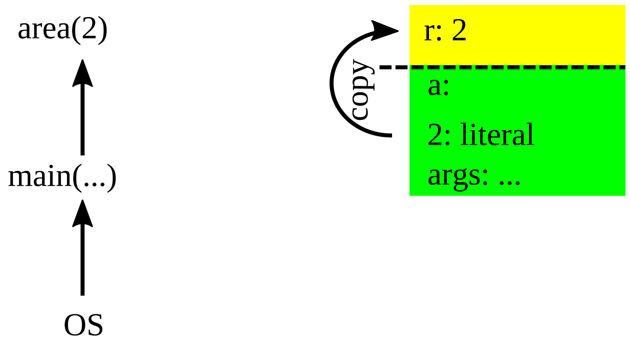
a:

2: literal

args: ...

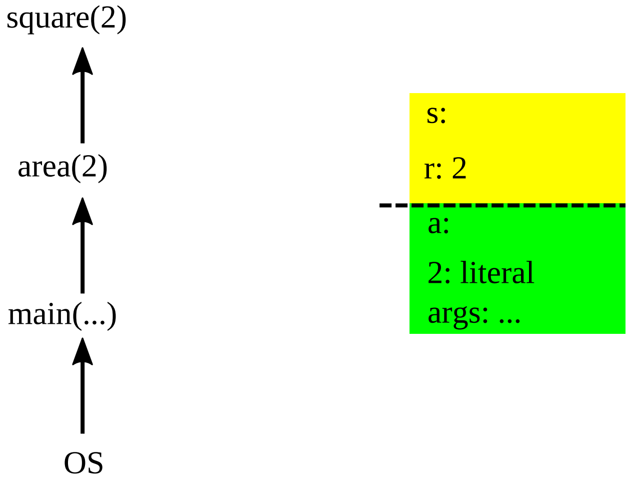
Call stack trace

```
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);
  }
}
```



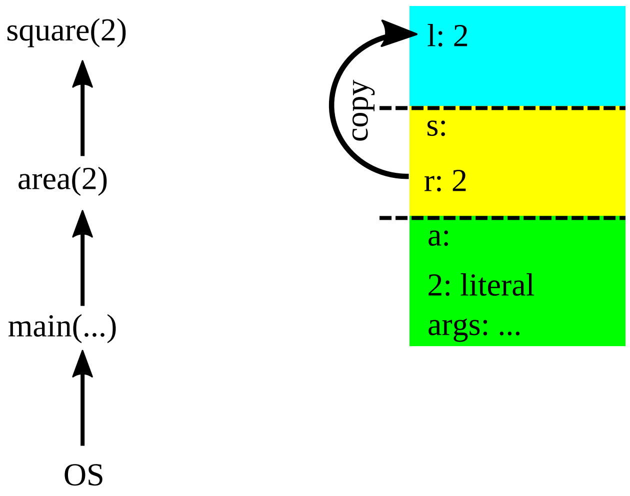
Call stack trace

```
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);
  }
}
```



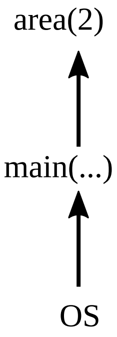
Call stack trace

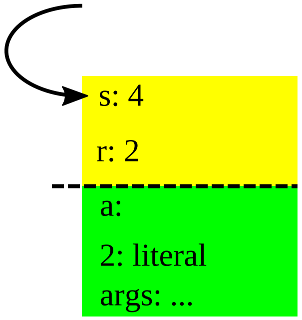
```
public class CallStackExample {
  static double square(double l){
    return l * l;
  }
  static double area(double r) {
    final double s = square(r);
    return Math.PI * s;
  }
  public ... main(String[] args){
    double a = area(2);
    System.out.println(
              "Area:" + a);
  }
}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



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

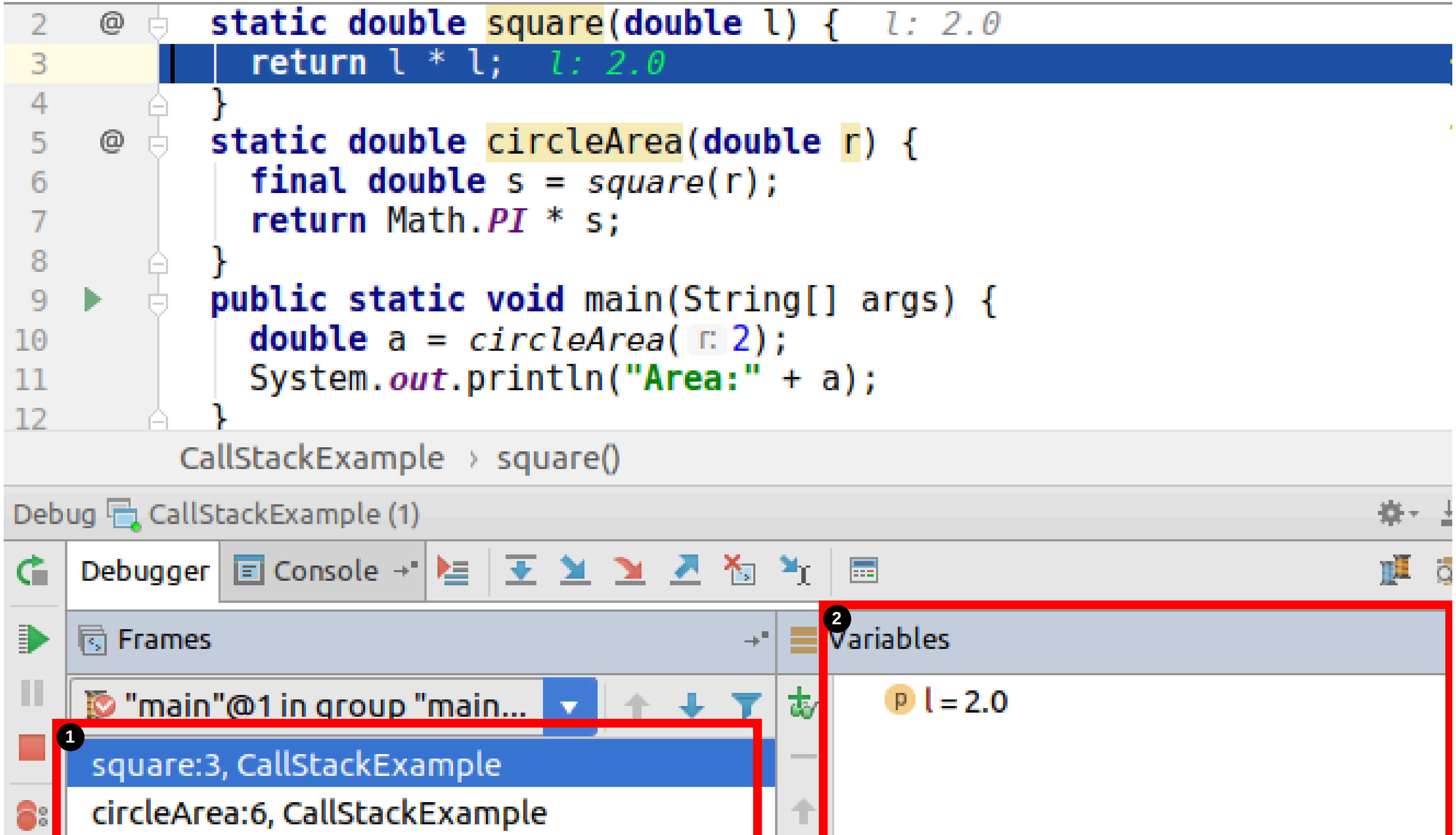
```
public class CallStackExample {
  static double square(double l){
    return l * l;
  }
  static double area(double r) {
    final double s = square(r);
    return Math.PI * s;
  }
  public ... main(String[] args){
    double a = area(2);
    System.out.println(
            "Area:" + a);
  }
}
```

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



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 `int` representation

```
public class Day {

  static public final int
    MONDAY    = 1,
    TUESDAY   = 2,
    WEDNESDAY = 3,
    THURSDAY  = 4,
    FRIDAY    = 5,
    SATURDAY  = 6,
    SUNDAY    = 7;
}
```

Weekly offered lectures using constants

```
Class Driver:

final Lecture
  phpIntro     = new Lecture(Day.MONDAY, "PHP introduction"),
  advancedJava = new Lecture(Day.FRIDAY, "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 introduction"),

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

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

```
Lecture «PHP introduction»
  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

```java
/**
* 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;
  }
}
```

```java
// 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(),
    FRIDAY    = 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 `getPrice()`

```
/**
 * 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.hdm_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; }
}
```

enum Day 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
  }
}
```

`enum` constructor being implicitly `private`

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

Compile time warning:

Modifier 'private' is redundant for `enum` constructors

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

Compile time error:

Modifier 'public' not allowed here

Prohibits `enum` external instance creation.

Related exercises

Exercise 123: Compass directions
Exercise 124: Compass direction neighbours

From https://www.urbandictionary.com

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 --message` "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
>
```

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

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

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 ❶(HEAD -> master)
Author: Martin Goik <goik@hdm-stuttgart.de>
Date:   Sat May 25 11:56:00 2019 +0200

    Removing duplicate headline ❷

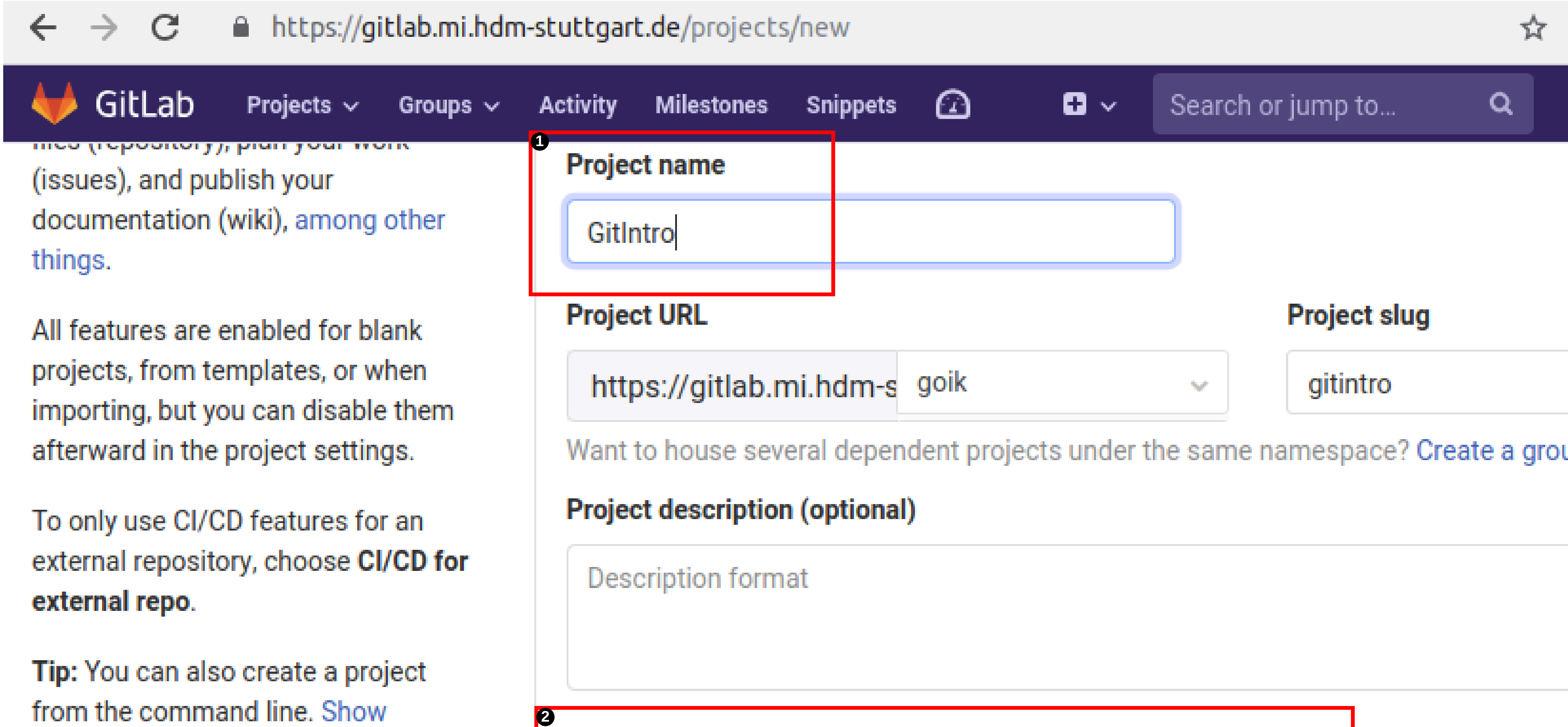
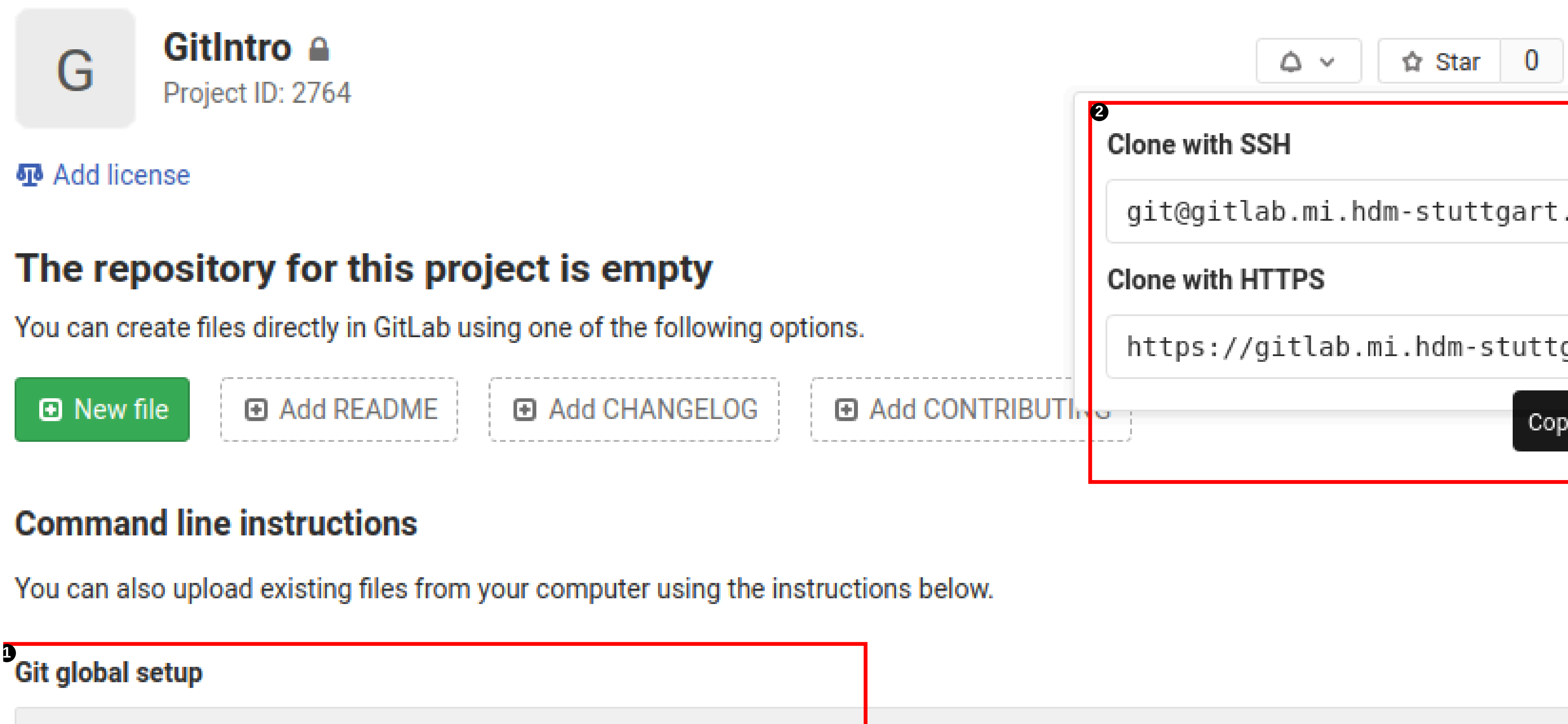
commit 7f119fac36e02e4c5a7f04f022217b6f744d6e1d ❸
Author: Martin Goik <goik@hdm-stuttgart.de>
Date:   Sat May 25 11:49:52 2019 +0200

    Project Readme.md ❹ ...
```

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.mi.hdm-stuttgart.de`.
2. Connect local repository to remote
3. push or pull content

Step 1: Create remote repository



Step 2: Retrieve remote repository address



Step 2: Connect to remote repository

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

❶ **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	User B
Edit: ... **Driver class** ...	-
git commit, git push	-
-	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.mi.hdm-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
++>>>>>>> dbbedb0fc29d77beeaaada37f2538d78f82bac93
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 poir
 */
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.mi.hdm-stuttgart.de/goik/gitintro.git
   10cf21c..173487a  master -> master
```

Related exercises

Exercise 126: git distributed, DIY