

MSc/ICY Software Workshop

Classes and Objects, JUnit Tests

Manfred Kerber www.cs.bham.ac.uk/~mmk

Tuesday 8 October 2019

Overview

- 1 Pocket calculator computations, base types, simple strings, variables, static methods, JavaDoc
Wed/Thu/Fri: 1st Lab Lecture (login, editor, javac, javadoc)
- 2 **Classes, objects, methods, JUnit tests**
Wed/Thu/Fri: 2nd Lab Lecture (Eclipse)
- 3 Conditionals, 'for' Loops, arrays, ArrayList
- 4 Exceptions, I/O (Input/Output)
- 5 Functions, interfaces
- 6 Sub-classes, inheritance, abstract classes
- 7 Inheritance (Cont'd), packages
- 8 Revision
- 9 Graphics
- 10 Graphical User Interfaces
- 11 Graphical User Interfaces (Cont'd)

Changes possible

Classes as Generalized Types

Classes can be considered as generalized types.

There are 8 basic **types** in Java (such as **int** and **double**).

Classes are general and can be user defined. For instance, we can define a class **Date**, consisting of an **int**, a **String**, and another **int**, representing the day of the month, the month, and the year.

Objects as Elements of Classes

Objects are elements of **Classes**.

E.g., **8 October 2019** is a **Date**.

Formally in Java

```
/** First, we declare the variables we use in this class.  
 *  *private* means that the variable cannot be accessed  
 *  from outside the class.  
 *  (As opposed to *public* which means that it can be  
 *  accessed. We declare the variables as private because  
 *  of data encapsulation.)  
 *  We do not declare variables that are not necessary,  
 *  since this can lead to all sorts of problems!  
 */  
public class Date{  
    private int    day;  
    private String month;  
    private int    year;  
}
```

Note: Each class goes in a separate file!

Formally in Java – Constructor

```
/** This constructor creates a date from the three parts:
 *  day, month, and year, which are an int, a String,
 *  and an int, respectively.
 *  @param d The day of the month (e.g., 8 in 8 October 2019)
 *  @param m The month in the year (e.g., "October" in 8 October 2019)
 *  @param y The year (e.g., 2019 in 8 October 2019)
 */
public Date (int    d,
             String m,
             int     y){
    day      = d;
    month    = m;
    year     = y;
}
```

Getter methods

```
/** Now we write *methods* to get the parts of a Date,
 * so called *accessor methods* or *getters*
 */

/**
 * @return The day of a Date (e.g., 8 from 8 October 2019)
 */
public int getDay(){
    return day;
}

/**
 * @return The month of a Date (e.g., "October" from 8 October 2019)
 */
public String getMonth(){
    return month;
}

/**
 * @return The year of a Date (e.g., 2019 from 8 October 2019)
 */
public int getYear(){
    return year;
}
```

Setter Methods

```
/** Now we write methods to set the parts of a Date,  
 * so called *setters*.  
 */  
  
/**  
 * sets the day of a Date  
 * @param newDay is the new day to which the day is set  
 */  
public void setDay(int newDay){  
    day = newDay;  
}  
  
/**  
 * sets the month of a Date  
 * @param newMonth is the new month to which the month is set  
 */  
public void setMonth(String newMonth){  
    month = newMonth;  
}
```

(Likewise for setYear.)

Printing of Objects by the toString Method

```
/**
 * this method says how to print a date
 * @return A String how the object is printed.
 */
public String toString(){
    return day + " " + month + " " + year;    // European
    //return year + ", " + month + " " + day; // American
}
```

Checking equality by the equals Method

```
/**
 * this method checks whether the date is equal to a
 * second date
 * @param date The second Date.
 * @return true if the current date (*this*) is equal
 *         to the date it is compared to, that is,
 *         if it agrees with it in day, month, and year.
 * NOTE: equality is a tricky concept!
 */
public boolean equals(Date date){
    return (this.getDay() == date.getDay()) &&
           (this.getMonth().equals(date.getMonth())) &&
           (this.getYear() == date.getYear());
}
```

Some boolean expressions

`3 == 4`

Some boolean expressions

`3 == 4`

\mapsto

`false`

Some boolean expressions

`3 == 4`

\mapsto

false

`3 > 4`

Some boolean expressions

`3 == 4`

\mapsto

`false`

`3 > 4`

\mapsto

`false`

Some boolean expressions

`3 == 4`

\mapsto

`false`

`3 > 4`

\mapsto

`false`

`3 < 4`

Some boolean expressions

`3 == 4`

\mapsto

false

`3 > 4`

\mapsto

false

`3 < 4`

\mapsto

true

Some boolean expressions

`3 == 4`

\mapsto

false

`3 > 4`

\mapsto

false

`3 < 4`

\mapsto

true

`3 < 4 && 4 < 5`

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>		

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>		

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>		

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>	\mapsto	false

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>	\mapsto	false
<code>"test".equals("test")</code>		

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>	\mapsto	false
<code>"test".equals("test")</code>	\mapsto	true

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>	\mapsto	false
<code>"test".equals("test")</code>	\mapsto	true
<code>"test1".equals("test2")</code>		

Some boolean expressions

<code>3 == 4</code>	\mapsto	false
<code>3 > 4</code>	\mapsto	false
<code>3 < 4</code>	\mapsto	true
<code>3 < 4 && 4 < 5</code>	\mapsto	true
<code>4 < 3 4 < 5</code>	\mapsto	true
<code>!(4 < 3 4 < 5)</code>	\mapsto	false
<code>(4 < 3 4 < 5) && 3 == 4</code>	\mapsto	false
<code>"test".equals("test")</code>	\mapsto	true
<code>"test1".equals("test2")</code>	\mapsto	false

Another EXAMPLE – BankAccount

```
/** BankAccount is a class for a very simple bank
 *  account created from a bank account and the
 *  name of the account holder.
 *  @author   Manfred Kerber
 *  @version  10 October 2018
 */
public class BankAccount{
    private int      accountNumber;
    private String   accountName;
    private int      balance;
```

Constructor

```
/** BankAccount is a constructor for a very
 *  simple bank account created
 *  @param  accountNumber is the account number as int
 *  @param  accountName the account name as String
 */
public BankAccount(int    accountNumber,
                    String accountName){
    this.accountNumber    = accountNumber;
    this.accountName      = accountName;
    this.balance           = 0;
}
```

A Second Constructor

```
/** BankAccount is a constructor for a very
 *  simple bank account created
 *  @param  accountNumber The account number as an int.
 *  @param  accountName The account name as a String.
 *  @param  balance The initial balance on the account as a
 */
public BankAccount(int    accountNumber,
                   String  accountName,
                   int     balance){
    this.accountNumber    = accountNumber;
    this.accountName      = accountName;
    this.balance           = balance;
}
```

Getter methods

```
/* Now we write methods to get the parts of a
 * BankAccount, so called accessor methods, the getters.
 */
/**
 * @return the account number of a
 * BankAccount as int
 */
public int getAccountNumber(){
    return accountNumber;
}

/**
 * @return the accountName as a String
 */
public String getAccountName(){
    return accountName;
}

/**
 * @return the balance of a BankAccount
 */
public int getBalance(){
    return balance;
}
```


Setter Methods

```
/* Now we write methods to set the parts of a bank account.  
 * so called setters.  
 */  
  
/**  
 * sets the account number of a BankAccount  
 * @param accountNumber for the changed account number  
 */  
public void setAccountNumber(int accountNumber){  
    this.accountNumber = accountNumber;  
}  
  
/**  
 * sets the balance of a BankAccount  
 * @param newBalance the new balance on the account  
 */  
public void setBalance(int balance){  
    this.balance = balance;  
}
```

Printing of Objects by the toString Method

```
/** toString defines how to print a BankAccount
 *
 * @return the print type of an account
 */
public String toString(){
    return "Account number: " + accountNumber +
        " Account name: " + accountName +
        " Balance: " + balance;
}
```

Checking equality by the equals Method

```
/**
 * this method checks whether the BankAccount is equal to
 * second BankAccount
 * @return true if the current BankAccount (*this*) is equal
 *         to the BankAccount it is compared to, that is,
 *         if it agrees with it in number, name, balance.
 * @param a The second BankAccount.
 * NOTE: equality is a tricky concept!
 */
public boolean equals(BankAccount a){
    return
        (this.getAccountNumber() == a.getAccountNumber()) &&
        (this.getAccountName().equals(a.getAccountName())) &&
        (this.getBalance() == a.getBalance());
}
```

Write comments in the following form

```
/**
 *   In the following we define the Date class ...
 *   @author Manfred Kerber
 *   @version 2018-10-10
 */
public class Date{
    /**
     *   toString of a Date gives a printed version of a Date
     *   @return The String how the date will be printed.
     */
    public String toString(){
        return day + " " + month + " " + year;
    }
}
```

javac vs javadoc

With javac we compile the .java file:

```
javac BankAccount.java
```

With javadoc we extract documentation from it:

```
javadoc -author -version BankAccount.java
```

We use the tags:

- `author` (author of a class)
- `version` (the date when class written, e.g.)
- `param` (one entry for each parameter)
- `return` (return value for non void methods)

JUnit Testing

In JUnit testing we compare the **expected result** of a method or a computation to the **actual result**. If the result agrees then the test **passes**, otherwise it **fails**.

We use initially only `assertEquals`, `assertFalse`, and `assertTrue`.

Details on <http://junit.org/>

For a fuller list of assertions see:

<https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html>

Running JUnit tests

- To run JUnit tests (Version 5), a so-called jar file with name `junit-platform-console-standalone-1.5.2.jar` is needed.
- Store the file in a directory of your choice, let us call it `DIRECTORY`. In the following replace `DIRECTORY` by the actual location of the directory such as `/usr/local/java/`.
- Compile the file to be tested by `javac -d bin JUnit.java`. The option `-d bin` means that the `JUnit.class` file will be written to the directory `bin`.
- Compile the test file by `javac -d bin -cp bin:DIRECTORY/junit-platform-console-standalone-1.5.2.jar JUnitTests.java`
- Run the tests by `java -jar DIRECTORY/junit-platform-console-standalone-1.5.2.jar -class-path bin -scan-class-path`

Note that the names `JUnit.java` and `JUnitTests.java` must match.

JUnit Testing

```
@Test
public void assertEqualsTestInt() {
    assertEquals(4, 2 * 2,
        "failure in assertEqualsTestInt: " +
        " expected 4 == 2 * 2");
}
```

```
@Test
public void assertEqualsTest2() {
    assertEquals(2.0, //expected
        2.1, //actual
        0.11, // tolerance >= |expected - actual|
        "failure in assertEqualsTest2: " +
        "expected and actual values differ");
}
```


JUnit Testing (Cont'd)

```
@Test
public void assertFalseTest() {
    assertFalse(3 == 4,
        "failure in assertFalseTest: " +
        " expected false but got true" );
}
```

```
@Test
public void assertTrueTest() {
    assertTrue(2 < 5,
        "failure in assertTrueTest: " +
        "expected true but got false");
}
```