# MSc/ICY Software Workshop
# Classes and Objects, JUnit Tests

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

5 October 2016



## 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., 5 October 2016 would be 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.
 */
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.
 */
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
 */
public int getDay(){
    return day;
}

/**
 * return the month of a Date
 */
public String getMonth(){
    return month;
}

/**
 * return the year of a Date
 */
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

```
public boolean equals(Date date){
    return (this.getDay() == date.getDay()) &&
           (this.getMonth().equals(date.getMonth())) &&
           (this.getYear() == date.getYear());
}
```



## Some boolean expressions

| | | |
|---|---|---|
| `3 == 4` | ↦ | false |
| `3 > 4` | ↦ | false |
| `3 < 4` | ↦ | true |
| `3 < 4 && 4 < 5` | ↦ | true |
| `4 < 3 \|\| 4 < 5` | ↦ | true |
| `!(4 < 3 \|\| 4 < 5)` | ↦ | false |
| `(4 < 3 \|\| 4 < 5) && 3 == 4` | ↦ | false |
| `"test".equals("test")` | ↦ | true |
| `"test1".equals("test2")` | ↦ | 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  6 October 2015
 */
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;
}
```



## Getter methods

```
/* Now we write methods to get the parts of a
 * BankAccount, so called accessor methods.
 */
/**
 *  @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

```
public boolean equals(Account a){
   return
     (this.getAccountNumber() == a.getAccountNumber()) &&
     (this.getAccountName().equals(a.getAccountName)) &&
     (this.getBalance() == a.getBalance());
}
```

## JavaDoc

Write comments in the following form

```
/**
 *    In the following we define the Date class ...
 *    @author Manfred Kerber
 *    @version 2015-10-07
 */
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://github.com/junit-team/junit/wiki/Assertions

Write the tests into a class with an appropriate name, e.g. Name.java, compile it with javac Name.java, and run it with java org.junit.runner.JUnitCore Name.



## JUnit Testing

```
@Test
  public void assertEqualsTest1() {
    assertEquals("failure in assertEqualsTest1: " +
     " expected string not equal given string",//errorMsg
     "text", //expected value
     "te" + "xt");//actual value
}

@Test
  public void assertEqualsTest2() {
    assertEquals("failure in assertEqualsTest2: " +
     " expected number not approx. equal given number",//errorMsg
     2.0,    //expectedValue
     2.1,    //actualValue
     0.11);  // tolerance >= |expectedValue - actualValue|
}
```



## JUnit Testing (Cont'd)

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

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