Java – basics at a glance

Java is a programming language for object-oriented programming. This means to solve technical problems by the use of classes. First one will model the entities in scope as Java classes, before one will solve the specific problems using these classes .

A class represents an entity of the area of concern. Each class has a unique name, a constructor (named as the class itself), specific attributes and specific functionality implemented as so-called methods

Each Java class will be saved in a source code file *ClassName*.java. In order to run a Java program there must be at least one class that implements a main-method using this signature:

public static void main(String[] args)

Classes are grouped into packages according to technical aspects.

Get started: variables and data types

A key aspect in programming are variables to store values. This is almost the same as in calculations in Math or Physics (e.g. v = a*t). But you need to know, that each Java variable is of a specific data type. Furthermore variables can also be used to store objects (instances) of classes.

```
int a = 12;        double money = -123.45;        boolean ok = false;        String name = "Sue";
```

```
Random rnd = null;
```
Declaration of the variable rnd as object of class Random. Assigning **null** to a variable states there is currently no value defined for this variable.

1 Example of a Java class

Source code file BottleCase.java – contains implementation of class BottleCase – represents a box to collect refundable bottles

Name of the package, the class is related to

Class definition

Definition class internal (private) variables; final is used to mark constants

Constructor: named equally to the class, creates instances of the class; no return data type information required; used to initialize internal variables

Method (action/function) of the class, example shows a parameter n without any return value (key word: void).

Method to show the calculation of a return-value (return v). The data type of the returned variable must be declared before the name of the method (e.g. double).

Main-method to start the program
- new creates an object of class BottleCase assigned to variable b
- calls the method addMore for object b
- shows the total value of BottleCase b

Output of the program is: EUR: 0.64

```java
package org.htwd.refund;

public class BottleCase {
    private final double BOTTLEVALUE = 0.08;
    private final int CAPACITY;
    private int count;

    public BottleCase(int capacity) {
        this.CAPACITY = capacity;
        this.count = 0;
    }

    public void addMore(int n) {
        this.count = this.count + n;
    }

    public double getTotalValue() {
        double v = this.BOTTLEVALUE *
                                    this.count;
        return v;
    }

    public static void main(String[] args) {
        BottleCase b = new BottleCase(20);
        b.addMore(8);
        System.out.println("EUR: " +
                    b.getTotalValue());
    }
}
```

Method calls are stated as: InstanceVariable.MethodName(Parameter)

2 Console input

Import the Scanner-class

The Scanner-class is used to read console input (System.in)

The Scanner-method nextInt reads an integer number into an integer-variable.

nextLine reads all characters until end of line (used to consume line break here).

nextLine reads all characters until end of line into a string-variable.

close the scanner

```java
import java.util.Scanner;

public static void main(String[] args) {
    BottleCase b = new BottleCase(20);
    b.addMore(3);
    System.out.println("Bottles to add?");
    Scanner sc = new Scanner(System.in);
    int num = sc.nextInt();
    sc.nextLine();

    b.addMore(num);
    System.out.println("We have total EUR: "
            + b.getTotalValue());

    System.out.println("What's your name?");
    String name = sc.nextLine();
    System.out.println("Great job " + name);
    sc.close();
}
```

3 Conditional execution

if compares values and branches accordingly. In case the condition given in round brackets yields true, the subsequent statements (enclosed in curly brackets) will be performed.

The **else**-part is only to be used in combination with a previous if-statement. It defines an alternative path of execution, in case the if-condition was not full-filled. The else-part is optional.

```java
public BottleCase(int capacity) {
    if (capacity > 0) {
        this.CAPACITY = capacity;
    } else {
        this.CAPACITY = 0;
        System.out.println("Capa. set to 0");
    }
    count = 0;
}

public void addMore(int n) {
    if (n > 0) {
        this.count = this.count + n;
    }
}
```

4 Loops

A **while**-loop repeats all the statements inside the loop body as long as the condition located inside the round brackets yields true.

As long as the value of variable people is greater than zero more bottles will be added to the BottleCase b.

At every execution of the loop the value stored in the variable people needs to be reduced by one. This is very important, otherwise the loop would never terminate.

```java
public static void main(String[] args) {
    BottleCase b = new BottleCase(20);
    int num = 0;

    Scanner sc = new Scanner(System.in);
    System.out.println("How many people
                                collected bottles?");
    int people = sc.nextInt();
    sc.nextLine();

    while (people > 0) {
        System.out.println("Number of bottles to add?");
        num = sc.nextInt();
        sc.nextLine();
        b.addMore(num);
        people = people - 1;
    }

    System.out.println("We have total EUR: "
            + b.getTotalValue());

    sc.close();
}
```

5 Arrays to store data

An array stores a fix number of data items of a dedicated data type. In order to access data stored in the array an index is used. An index is an integer number that refers the position of the desired array item. The first item is related to index zero.

The variable cases represents an array of type BottleCase. The array has a size of 3 items.

cases.length is the length of the array.

Store the BottleCase object b at index position i in the array cases.

Iterate the array using the index variable i.

Assignment of a value to variable b (BottleCase) by reading the value from index position i in the array cases.

```java
public static void main(String[] args) {
    int count;
    Scanner sc = new Scanner(System.in);

    BottleCase b;
    BottleCase[] cases = new BottleCase[3];

    int i = 0;
    while (i < cases.length) {
        b = new BottleCase(20);
        System.out.println("How many bottles to add?");
        count = sc.nextInt();
        sc.nextLine();
        b.addMore(count);
        cases[i] = b;
        i = i + 1;
    }
    sc.close();

    double sum = 0.0;
    i = 0;
    while (i < cases.length) {
        b = cases[i];
        sum = sum + b.getTotalValue();
        i = i + 1;
    }

    System.out.println("We have total EUR: " + sum);
}
```

6 Make use of objects

This example of a Java-Program illustrates, how to use the classes BottleCase and RefundStation to calculate the bottle refund by the interaction of both classes.

Both classes are located in the package org.htwd.refund. The implementation of the class RefundStation refers to the class BottleCase – watch the implementation of the method returnCase.

Finally the main-method of he class BottleRefundDemo shows the calculation of the bottle refund. It creates a random count of BottleCase objects. They are filled with a random count of bottles. The class RefundStation calculates and shows the refund for all these bottle cases.

```java
package org.htwd.refund;

public class BottleCase {
    private double BOTTLEVALUE = 0.08;
    private final int CAPACITY;
    private int count;

    public BottleCase(int capacity) {
        this.CAPACITY = capacity;
        this.count = 0;
    }

    public void addMore(int n) {
        if (n > 0 && (this.count+n) <= this.CAPACITY) {
            this.count = this.count + n;
        }
    }

    public double getTotalValue() {
        return this.BOTTLEVALUE * this.count;
    }
}
```

```java
package org.htwd.refund;

public class RefundStation {
    private int collectedCases;
    private double sumRefund;

    public RefundStation() {
        this.collectedCases = 0;
        this.sumRefund = 0.0;
    }

    public double returnCase(BottleCase c) {
        this.collectedCases++;
        double val = c.getTotalValue();
        this.sumRefund = this.sumRefund + val;
        return val;
    }

    public void printStatus() {
        System.out.println("collected Cases:   "
                            + this.collectedCases);
        System.out.println("Sum of refund EUR: "
                                + this.sumRefund);
    }
}
```

Create an object of a random number generator to draw a random number n in the range of 1-10.

Initialize an object variable b of type BottleCase to null – so initially it does not reference an object.

Create an array cases to store n BottleCase objects.

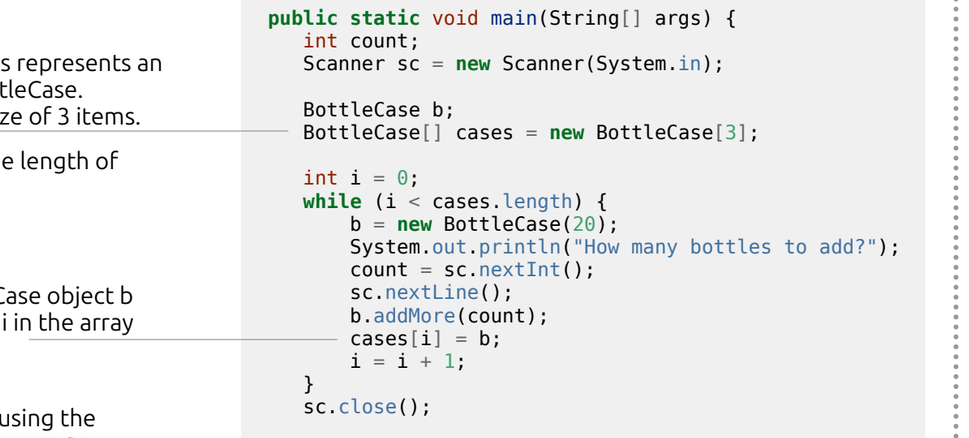
Loop to repeat the creation of BottleCase objects, to add empty bottles and to store them at index position I in the array.

Create the object of a RefundStation and show its status.

Iterate all BottleCases in the array cases, store each case into variable c and perform the return of the case at the RefundStation-object r.

```java
package org.htwd.refund;
import java.util.Random;

public class BottleRefundDemo {

    public static void main(String[] args) {
        Random rnd = new Random();
        int n = rnd.nextInt(10)+1;
        int number;
        int i = 0;

        BottleCase b = null;
        // create BottleCases in an array
        BottleCase[] cases = new BottleCase[n];
        while ( i < cases.length ) {
            b = new BottleCase(20);
            number = rnd.nextInt(15)+1;
            b.addMore(number);
            cases[i] = b;
            i++;
        }

        // process all the BottleCases
        // at the RefundStation
        RefundStation r = new RefundStation();
        r.printStatus();
        i = 0;
        BottleCase c = null;
        while ( i < cases.length ) {
            c = cases[i];
            r.returnCase(c);
            i++;
        }
        r.printStatus();
    }
}
```

Integrated development environment: IntelliJ https://www.jetbrains.com/idea/



In order to obtain an understanding of a program it is essential to be know what class an object variable is related to. Because the class of an object variable determines what methods (functionality) one could call or perform for this variable.

Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences
Contact: robert.ringel@htw-dresden.de