ETH zürich

Assignment 3

Felix Friedrich, Fabian Stutz, Lars Widmer

TA lecture, Informatics II D-BAUG

March 11, 2014

Presence Hours

The second set of **presence hours (Präsenzstunden)** will take place today.

Guided Programming

- Day: Every Thursday
- Room: **HIL E 15.2**
- Time: **15:00-18:00**
- Assistant: Timon Gehr

The room **changes** over the semester.
You find the list of rooms on the course website.

In the e-mail (sent out to all students) the dates of Mondays have been mentioned. That was wrong! Apologies, it's always on Thursdays!

Outline

Classes vs. Objects

Question

What's the difference between classes and objects?

Classes vs. Objects

What's the difference between classes and objects?

- The programmer writes classes not objects.

Classes vs. Objects

What's the difference between classes and objects?

- The programmer writes classes not objects.
- The class represents a construction plan e. g. printing plate for a book.

Classes vs. Objects

What's the difference between classes and objects?

- The programmer writes classes not objects.
- The class represents a construction plan e. g. printing plate for a book.
- An object represents such a book.

Classes vs. Objects

What's the difference between classes and objects?

- The programmer writes classes not objects.
- The class represents a construction plan e. g. printing plate for a book.
- An object represents such a book.
- There can be multiple books printed with the same plate.

Classes vs. Objects

What's the difference between classes and objects?

- The programmer writes classes not objects.
- The class represents a construction plan e. g. printing plate for a book.
- An object represents such a book.
- There can be multiple books printed with the same plate.
- The programmer can request objects of a class. The construction is done by the computer.

Classes vs. Objects

What's the difference between classes and objects?

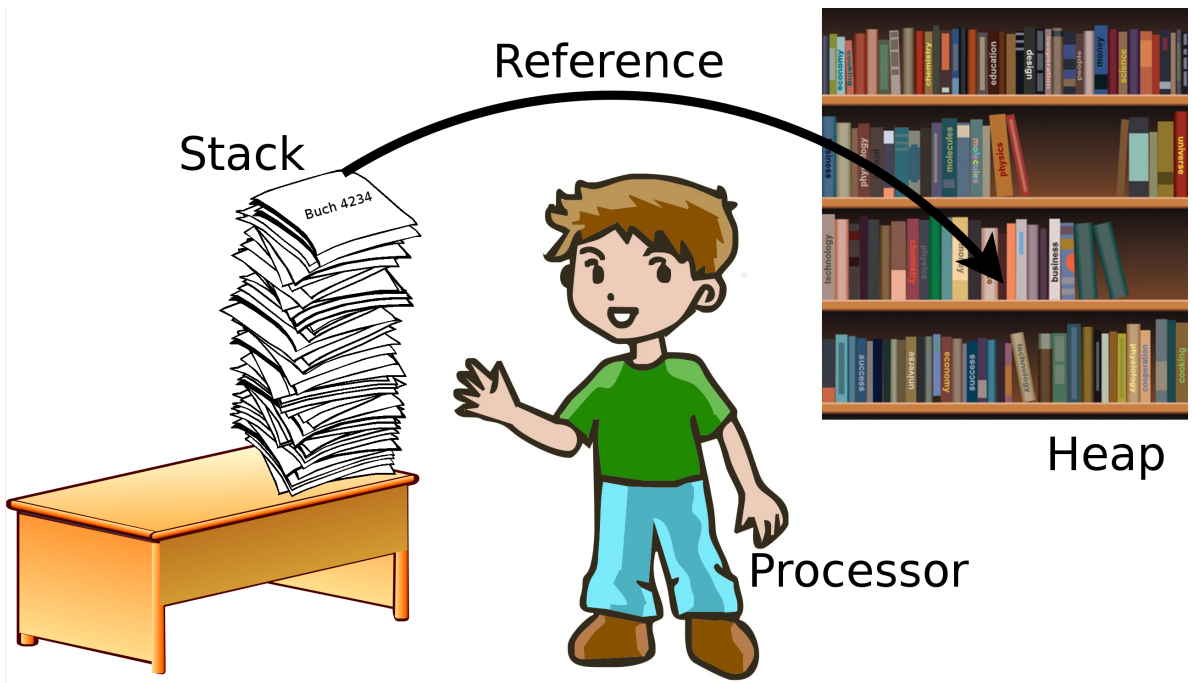
- The programmer writes classes not objects.
- The class represents a construction plan e. g. printing plate for a book.
- An object represents such a book.
- There can be multiple books printed with the same plate.
- The programmer can request objects of a class. The construction is done by the computer.
- The convention suggests **class-names** to start with an **uppercase** letter and object-names to start with a lowercase letter.

Outline

Repetition: If you're a Processor

Question:

Where are objects located and where are the references?



Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.
- If you pile 100 books on your desk, you can't work anymore.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.
- If you pile 100 books on your desk, you can't work anymore.
- Therefore only single sheets of papers go to your **stack**.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.
- If you pile 100 books on your desk, you can't work anymore.
- Therefore only single sheets of papers go to your **stack**.
- Books you place in your bookshelf.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.
- If you pile 100 books on your desk, you can't work anymore.
- Therefore only single sheets of papers go to your **stack**.
- Books you place in your bookshelf.
- So to speak, your bookshelf is your **heap** memory.

Repetition: If you're a Processor

As an analogy you may think of *your room* being a **computer** while *you* as a **processor** are doing your *homework* which is the **program**:

- Your **stack** is a pile of sheets of paper on your desk.
- An **object** could be a book.
- If you pile 100 books on your desk, you can't work anymore.
- Therefore only single sheets of papers go to your **stack**.
- Books you place in your bookshelf.
- So to speak, your bookshelf is your **heap** memory.
- You may place a note on a piece of paper about which book your using for what purpose. This note is a **reference**. The **reference** is kept in the **stack**, but it points to the **heap**.

Class Ingredients

Question

Please name the parts a class consists of.

Class Ingredients: Basic List

1. Access modifier

Class Ingredients: Basic List

1. Access modifier
2. Name

Class Ingredients: Basic List

1. Access modifier
2. Name
3. Constants (similar to a variable but immutable)

Class Ingredients: Basic List

1. Access modifier
2. Name
3. Constants (similar to a variable but immutable)
4. Variables (called **fields** or **properties**)

Class Ingredients: Basic List

1. Access modifier
2. Name
3. Constants (similar to a variable but immutable)
4. Variables (called **fields** or **properties**)
5. Methods (replacing procedures and functions in other languages)

Class Ingredients

Question

There is a special method which always bears the same name as the class itself. This method is automatically executed, when a new object of the class is created. What's that method?

Class Ingredient: Constructor

- Always has the **same** name as the class

Class Ingredient: Constructor

- Always has the **same** name as the class
- Is **automatically executed** when an object of the class is created

Class Ingredient: Constructor

- Always has the **same** name as the class
- Is **automatically executed** when an object of the class is created
- The constructor returns the reference to a new object of the class. Since this is always the case, java is doing this behind the scenes. You even don't have to give a return type.

Class Ingredient: Constructor

- Always has the **same** name as the class
- Is **automatically executed** when an object of the class is created
- The constructor returns the reference to a new object of the class. Since this is always the case, java is doing this behind the scenes. You even don't have to give a return type.
- The **main usage** of the constructor is to **initialize** the fields (variables) of the class.

Class Ingredient: Constructor

- Always has the **same** name as the class
- Is **automatically executed** when an object of the class is created
- The constructor returns the reference to a new object of the class. Since this is always the case, java is doing this behind the scenes. You even don't have to give a return type.
- The **main usage** of the constructor is to **initialize** the fields (variables) of the class.
- The constructor can use **parameters**. This often comes in very handy.

Class Ingredient: Constructor

- Always has the **same** name as the class
- Is **automatically executed** when an object of the class is created
- The constructor returns the reference to a new object of the class. Since this is always the case, java is doing this behind the scenes. You even don't have to give a return type.
- The **main usage** of the constructor is to **initialize** the fields (variables) of the class.
- The constructor can use **parameters**. This often comes in very handy.
- If you don't specify a constructor, java uses an empty default constructor.

Classes are “between” Methods & Programs

It’s like a Matrjoschka*

1. A **method** contains **code**.
2. A **class** contains **methods**.
3. A **program** contains **classes**.

$$code \in method \in class \in program$$

*__Matrjoschkas__ are wrongly named as Babuschkas. Babuschka actually means grandmother.

Outline

How much do you already know?

Question?

What is encapsulation?

What is Encapsulation

- The process of packing **data** and the **according methods** in one class (“*construction plan*”)

What is Encapsulation

- The process of packing **data** and the **according methods** in one class ("*construction plan*")
- A class therefore represents an individual **piece of functionality**

What is Encapsulation

- The process of packing **data** and the **according methods** in one class ("*construction plan*")
- A class therefore represents an individual **piece of functionality**
- It will be used to separate the `HiLo`-game from the `highscore`-handling

What is Encapsulation

- The process of packing **data** and the **according methods** in one class ("*construction plan*")
- A class therefore represents an individual **piece of functionality**
- It will be used to separate the `HiLo`-game from the `highscore`-handling
- Thus: **Separation of concerns**

Encapsulation Example

We have an example-program, which “invents” good passwords.

```
1 svYiS
2 W.avY.
3 *Fr_$Hp
4 EHscumjW
5 QKCsmlQqi
```

The program has the following methods

- `initDefaultChars()`
  Adds the normal letters uppercase and lowercase into a `LinkedList`.

The program has the following methods

- `initDefaultChars()`
  Adds the normal letters uppercase and lowercase into a `LinkedList`.
- `addChar(char ch)`
  Adds a user defined special character to the list (e. g. `'*'`)

The program has the following methods

- `initDefaultChars()`
  Adds the normal letters uppercase and lowercase into a `LinkedList`.
- `addChar(char ch)`
  Adds a user defined special character to the list (e. g. `'*'`)
- `getNewPassword(int length)`
  Returns a new password of the requested length. By randomly selecting characters from the `LinkedList`.

The program has the following methods

- `initDefaultChars()`
  Adds the normal letters uppercase and lowercase into a `LinkedList`.
- `addChar(char ch)`
  Adds a user defined special character to the list (e. g. `'*'`)
- `getNewPassword(int length)`
  Returns a new password of the requested length. By randomly selecting characters from the `LinkedList`.
- `main(String[] args)`
  Uses `getNewPassword(int length)` in a loop to print 5 random passwords with 5 different lengths.

Example Code, just if you're interested

```
1 public static String getNewPassword(int length)
2 {    String pw = "";
3      for (int i=0; i<length; ++i)
4      {    pw += buchstaben.get(
5                 rand.nextInt(buchstaben.size()));
6      }
7      return pw;
8 }
```

Main-Program

```
1  public static void main(String[] args)
2  {    initDefaultChars();
3       addChar('*');
4       addChar('/');
5       addChar('$');
6       addChar('.');
7       addChar('-');
8       addChar('_');
9       for (int i=0; i<5; ++i)
10      {    System.out.println(getNewPassword(5+i));
11      }
12 }
```

Wrap the functionality

1. Class `RandomPasswords`
   - `initDefaultChars()` → **Constructor**
   - `addChar(char ch)`
   - `getNewPassword(int length)` → `getNew(int length)`
2. `main(String[] args)`
   Creates an object e. g. `rps` of class `RandomPasswords` and uses the functionality through this object.

New Class Structure

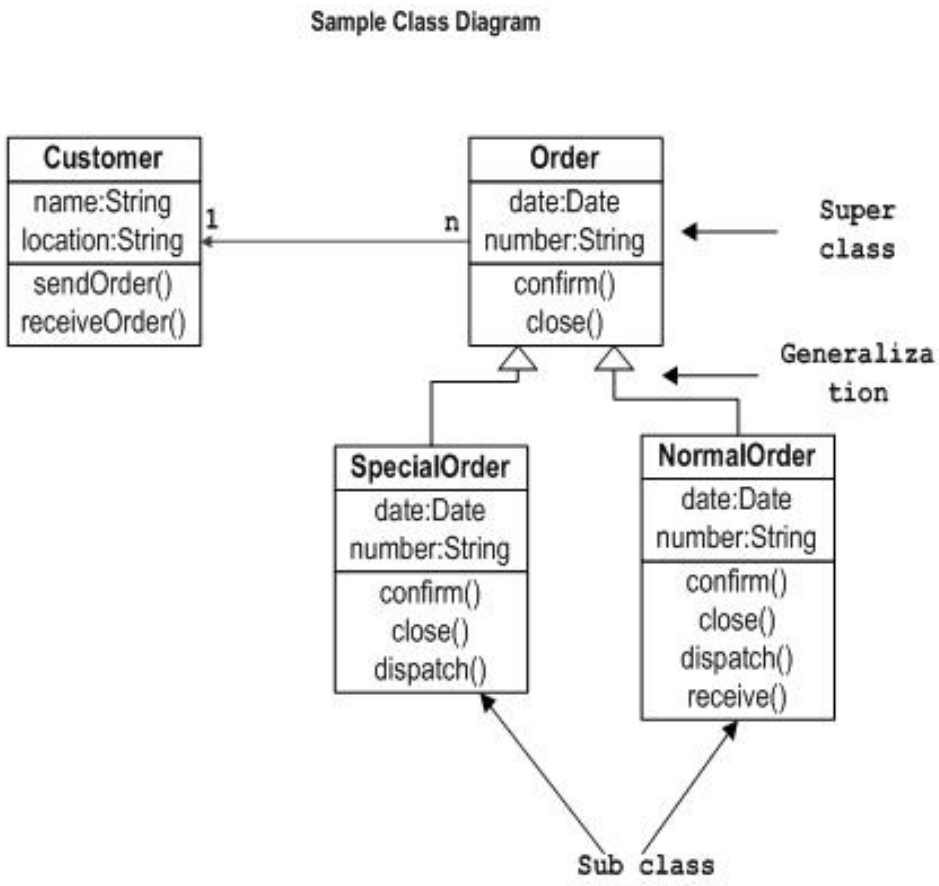
```
1 public class RandomPasswords
2 {    public RandomPasswords() { ... } // Constructor
3      public void addChar(char ch) { ... }
4      public String getNew(int length) { ... }
5 }
```

No more `static` keywords in the new class. There are no more substitutes for Pascal like procedures. We now have truly object-oriented methods which are encapsulated in a `class`.

New Main-Program

```
1  public static void main(String[] args)
2  {    RandomPasswords rps = new RandomPasswords();
3       rps.addChar('*'); // adding some
4       rps.addChar('/'); // special characters
5       rps.addChar('-'); // for getting
6       rps.addChar('_'); // safer passwords
7       rps.addChar('.'); // ...
8       rps.addChar('$'); // ...
9       for (int i=0; i<5; ++i)
10      {    System.out.println(rps.getNew(i+5));
11      }
12 }
```

Designing Classes



source: www.f5systems.in

Forming classes is not only important when *refactoring existing code*. Also when **planning a fresh program** from scratch, we structure it into classes.

Outline

Multiply Code

1. Ensure that vector length is equal to matrix height

Multiply Code

1. Ensure that vector length is equal to matrix height
2. Prepare empty result vector

Multiply Code

1. Ensure that vector length is equal to matrix height
2. Prepare empty result vector
3. For every row of the matrix ...

Multiply Code

1. Ensure that vector length is equal to matrix height
2. Prepare empty result vector
3. For every row of the matrix ...
   1. Set initial value 0 in the output vector at the current position

Multiply Code

1. Ensure that vector length is equal to matrix height
2. Prepare empty result vector
3. For every row of the matrix ...
   1. Set initial value 0 in the output vector at the current position
   2. For every element of the vector ...

Multiply Code

1. Ensure that vector length is equal to matrix height
2. Prepare empty result vector
3. For every row of the matrix ...
   1. Set initial value 0 in the output vector at the current position
   2. For every element of the vector ...
      1. Add the product of the according vector and matrix values to the current element in the result vector

Method Multiply

```
1  public static double[] multiply(
2          double[] v, double[][] m) {
3      assert (v.length == m.length);
4      double[] res = new double[m.length];
5      for (int j=0; j < m[0].length; ++j) {
6          res[j] = 0; // initial value
7          for (int i=0; i<v.length; ++i)
8              res[j] += m[i][j] * v[i];
9      }
10     return res;
11 }
```

Resulting Vector

The given **vector converges** to the following numbers:

1. 0.3252032520325203
2. 0.2274121797821752
3. 0.1254026691210308 7
4. 0.1436570026077619 6
5. 0.1783248964565117 8

Values bigger than 1

With a Value bigger than one . . .

There is **no** convergence anymore. The vector “explodes”. The numbers get higher and higher until they reach the **error state** “`naN`” (not a Number).

A line of 0s

With a line of zeros . . .

There is **no** convergence as well. All the vector values drop and finally stay at zero.

Why no convergence

Question?

Why de we lose convergence?
What's the reason?

What's the Reason?

Explanation

The values in the matrix are defined to be probabilities.
A probability has to be a value between zero and one:
$0 \leq p \leq 1$
If a line contains only zeros, it means that the probabilities are zero. Which means the surfer never leaves the page.

Outline

Fields & Constructor

```
1 private int[] highscore;
2 private int size;
3
4 public Highscore(int s) {
5     size = s;
6     highscore = new int[size];
7 }
```

As it should be, the constructor initializes the fields of the class.

Method `showHighscore`

```
1 public static void showHighscore()
2 {    System.out.println("*** HIGHSCORE ***");
3      for (int i=0; i<size; ++i)
4      {    System.out.println(highscore[i]);
5      }
6 }
```

Simple, isn't it?

HowTo `insertScore`?

1. Check if the score is good enough for the highscore

Algorithms like that are not easy to implement. There's a lot of potential for mistakes which produces errors under certain conditions. To be honest, writing something like that takes a good mixture of smart thinking **and** testing. Only one of them, usually is not enough.

HowTo `insertScore`?

1. Check if the score is good enough for the highscore
2. Locate position in highscore (while loop)

Algorithms like that are not easy to implement. There's a lot of potential for mistakes which produces errors under certain conditions. To be honest, writing something like that takes a good mixture of smart thinking **and** testing. Only one of them, usually is not enough.

HowTo `insertScore`?

1. Check if the score is good enough for the highscore
2. Locate position in highscore (while loop)
3. Move the lower scores one slot down

Algorithms like that are not easy to implement. There's a lot of potential for mistakes which produces errors under certain conditions. To be honest, writing something like that takes a good mixture of smart thinking **and** testing. Only one of them, usually is not enough.

HowTo `insertScore`?

1. Check if the score is good enough for the highscore
2. Locate position in highscore (while loop)
3. Move the lower scores one slot down
4. Insert the new score

Algorithms like that are not easy to implement. There's a lot of potential for mistakes which produces errors under certain conditions. To be honest, writing something like that takes a good mixture of smart thinking **and** testing. Only one of them, usually is not enough.

Method `insertScore`

```
1  public static void insertScore(int score)
2  {    // if good enough for a highscore:
3       if (score > highscore[size-1]) // 9
4       {    int pos = 0;
5            while (score < highscore[pos])
6            {    ++pos;
7            }
8            for (int i=size-2; i>=pos; --i)
9            {    highscore[i+1] = highscore[i];
10           }
11           highscore[pos] = score;
12      }
13 }
```

Questions?

Please

- Questions?
- Feedback?
- Wishes?
- Remarks?
- ...

We Wish You Success!