Assignment 4

Felix Friedrich, Lars Widmer, Fabian Stutz
TA lecture, Informatics II D-BAUG
March 11, 2014

Presence Hours Today (Präsenzstunden)

Not in the same room as last week

- **HIL C 29**
- **15:00 - 18:00**
- Timon Gehr *(arriving 15:45)*

Outline

Private Fields

Question

From where can a private variable (**field**) be accessed?

```
1 public class Testi
2 {   private int x;
3 }
```


Private Fields

Answer

From where can a private variable (**field**) be accessed?

- Just **inside** the **class**.
- If you create an object of this class, you can't access the field on the object.
- The field is **hidden** inside of the object.

```
1 public void main(String[] args)
2 {   Testi t;
3     t.x = 5; // DOESN'T WORK
4 }
```


Public Fields

Question

From where can a public variable (**field**) be accessed?

```
1 public class Dati
2 {   public int amount;
3 }
```


Public Fields

Answer

From where can a public variable (**field**) be accessed?

- From **everywhere**.
- If you create an object of this class, you can access the field on the object.

```
1 public void main(String[] args)
2 {   Dati d;
3     d.amount = 5; // WORKS!
4 }
```


Fields and Methods

Important

- Visibility is the same for methods and fields.
- Methods can be private, the same way as variables.
- In general it's **good practice** to keep as many things private as possible.

```
1 public class Testi
2 {   public int getAmount() { ... }
3     private void helper() { ... }
4     ...
5 }
```

ETH zürich

Outline

ETH zürich

What's the Computational Complexity?

```
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 }
```


ETH zürich

Computational Complexity of `insertScore`

Complexity

- Worst case we need to go through the full array two times.
- Thus the complexity is $O(2n)$.
- $O(2n) = O(n)$.

ETH zürich

What's the Computational Complexity?

```
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 }
```


ETH zürich

Computational Complexity of `multiply`

Complexity

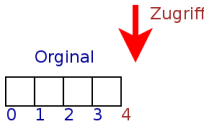
- If we assume the matrix size to be $x \times y$.
- The vector size is also x.
- The complexity is $O(x \times y)$.

ETH zürich

Outline

ETH zürich

Array Out-of-Bounds-Access

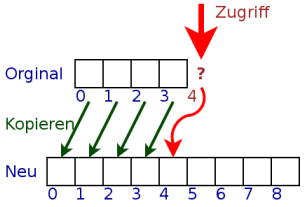


Java throws an `exception`.
Without any special handling, the program **crashes**.

ETH zürich

Array Out-of-Bounds-Access

To prevent this, we quickly create a new array which has enough space for the required element.

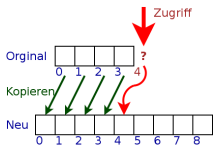


Outline

Array Out-of-Bounds-Access

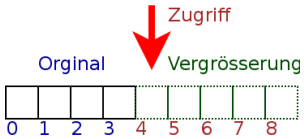
At first check if required index is valid.

1. If ok, access array
2. If not:
   1. Create a two times larger array
   2. Copy old array into new array
   3. Access the required element



Array Out-of-Bounds-Access

By **hiding** all this in a class, an out-of-bounds-access results in a magical increase in size.



The user doesn't have to even think about the array-size.

Outline

HiLo with `Player`

```
1 public class Player
2 {   public final String name;
3     public int score;
4
5     ...
6 }
```

The class encapsulates the `name` of the player together with its `score`.

HiLo with `Player`

I decided to **overload** the constructor.
Overloading is defining a method more than once but with different parameters.
Creating a new `Player`-object with given name and score in one go is great for testing.

```
1 public Player(String n)
2 {   name = n;
3     score = 0;
4 }
5 public Player(String n, int s)
6 {   name = n;
7     score = s;
8 }
```


HiLo with `Player`

If you do `System.out.println(p)` with a `Player` object `p`. It prints some object properties but neither `name` nor `score`.

Internally it calls `p.toString()`.

The method `toString` is always defined by default. If we overwrite it with something more useful, we can make it print `name` and `score`.

```
1 public String toString()
2 {   return name+ ':' + '␣' + score;
3 }
```


Outline

HiLo with `Player`

```
1 public Highscore(int s)
2 {   // size = s; // use highscore.length
3     highscore = new Player[s];
4     for (int i=0; i<s; ++i)
5     {   highscore[i] = new Player("<empty>",0);
6     }
7 }
```

Thanks to the overloaded constructor, prefilling the highscore is very simple.

HiLo with `Player`

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

Thanks to the overwritten `toString`-method, printing the Highscore list is as simple as before.

Outline

HiLo with `Player`

We additionally ask for the players name.

```
1 ...
2 System.out.println("Please enter your name: ");
3 String name = sc.next();
4 while (!done)
5 {   Player p = new Player(name);
6     ...
```

Creating a new object here is `very important`. Reusing it for another round would create a mess because it's still used by highscore.

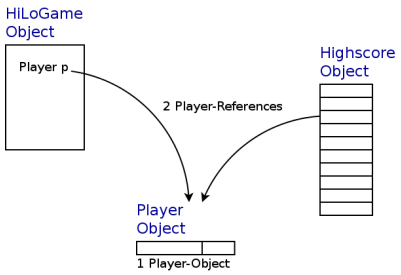
Having the `name` as a `final` field, fortunately enforces the recreation of the object if a new name would be used.

Reusing Objects

```
1 ...
2 while (!done)
3 {   p.score = 0; // Wrong!
4     ...
5     hs.insert(p);
6     ...
```

If after one round, we save the object into the highscore but reuse the object, the next round will change the old result in the highscore because it's still the **same object**, it's just referred from **two different places**.

Reusing Objects



Core Issue

```
1 Player p1 = new Player("Jack",0);
2 Player p2 = p1;
3 p2.score = 100;
4 System.out.println(p1.score);
```

Question

What's the output?

Core Issue

Answer

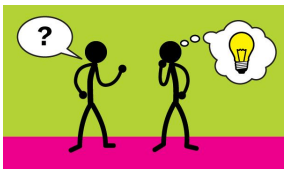
- Again, there are two references ...
- ... But only one (shared) object!
- Through both objects we modify the "one and only" object.
- The command line output is 100.

```
1 Player p1 = new Player("Jack",0);
2 Player p2 = p1;
3 p2.score = 100;
4 System.out.println(p1.score);
```

Questions from your side?

Please

- Feedback?
- Additions?
- Remarks?
- Wishes?
- ...



All the Best!