ETH zürich

Assignment 6

Felix Friedrich, Lars Widmer, Urs Müller, Severin Wischmann
TA lecture, Informatics II D-BAUG

March 27, 2014

“Präsenzstunden” Today

In the same room as last week *(standard)*

- **HIL E 15.2**
- **15:00 - 18:00**
- **Timon Gehr** *(arriving 15:45)*

We proudly present . . .

TA lectures in **Italian**

- **Thursdays**
- **12:45-13:30**
- **HIT F 31.1**
- **Raffaele Lauro**

. . . *Buon divertimento!*

Outline

Java Generics

“Type Parameters”

- Generics allow a type or method to operate on objects of various types.

Java Generics

“Type Parameters”

- Generics allow a type or method to operate on objects of various types.
- You can see **generics** as a **parameter** to hand a **type** to a class when instantiating objects of this class.

Java Generics

“Type Parameters”

- Generics allow a type or method to operate on objects of various types.
- You can see **generics** as a **parameter** to hand a **type** to a class when instantiating objects of this class.
- In the class there’s just place holder where in the object the type **must** be known (i. e. the place holder has been replaced by a type).

Java Generics

Building Houses

In the example of classes being a **construction plan** for a house while the houses are the objects built after the plan. The “generic” could e. g. be the color of the walls in the bathroom.

Java Generics

Building Houses

In the example of classes being a **construction plan** for a house while the houses are the objects built after the plan. The “generic” could e. g. be the color of the walls in the bathroom.
You can use the **same construction plan** and just “**replace**” the bathroom color from e. g. white to light-green.

Java Generics

“Type Parameters”

- Thanks to **generics** there are many helpful data structures *(implemented as classes)* which you can aggreagte **and while doing so**, set the according data type.

Java Generics

“Type Parameters”

- Thanks to **generics** there are many helpful data structures *(implemented as classes)* which you can aggreagte **and while doing so**, set the according data type.
- **Examples** are `ArrayList`, `Vector` and `LinkedList`.

Outline

Place Holder “★”

```
1  public class Example
2  {    private String bla;
3
4       public Example(String def)
5       {    bla = def;
6       }
7
8       public String get()
9       {    return bla;
10      }
11      public void set(String str)
12      {    bla = str;
13      }
14 }
```

Place Holder Wanted

The aim of generics, is to replace e. g. the type “String” by a placeholder “★”.

Place Holder “★”

Basically . . .

- **From:** `private String bla;`

Place Holder “★”

Basically . . .

- **From:** `private String bla;`
- **To:** `private ★ bla;`

Place Holder “★”

Basically ...

- **From:** `private String bla;`
- **To:** `private ★ bla;`
- **From:** `public String get()`

Place Holder “★”

Basically ...

- **From:** `private String bla;`
- **To:** `private ★ bla;`
- **From:** `public String get()`
- **To:** `public ★ get()`

Place Holder "★"

Basically ...

- **From:** `private String bla;`
- **To:** `private ★ bla;`
- **From:** `public String get()`
- **To:** `public ★ get()`
- **From:** `public void set(String str)`

Place Holder "★"

Basically ...

- **From:** `private String bla;`
- **To:** `private ★ bla;`
- **From:** `public String get()`
- **To:** `public ★ get()`
- **From:** `public void set(String str)`
- **To:** `public void set(★ str)`

Side by Side

```java
public class Example
{   private String bla;

    public Example(String def)
    {   bla = def;
    }

    public String get()
    {   return bla;
    }

    public void set(String str)
    {   bla = str;
    }
}
```

```java
public class Example<T>
{   private T bla;

    public Example(T def)
    {   bla = def;
    }

    public T get()
    {   return bla;
    }

    public void set(T str)
    {   bla = str;
    }
}
```

Question

Who can see the five differences?

Example as a Generic

```
1  public class Example<T>
2  {    private T bla;
3
4       public Example(T def)
5       {    bla = def;
6       }
7
8       public T get()
9       {    return bla;
10      }
11      public void set(T str)
12      {    bla = str;
13      }
14 }
```

With Generics

"T" is the **place holder**.

Example as a Generic

```
1  public class Example<T>
2  {    private T bla;
3
4       public Example(T def)
5       {    bla = def;
6       }
7
8       public T get()
9       {    return bla;
10      }
11      public void set(T str)
12      {    bla = str;
13      }
14 }
```

With Generics

"T" is the **place holder**. When an object is built according to class `Example`, also the class place holder "T" has to be replaced by an object of an actual class.

Using a Generic Class

```
1 public class Test1 // Without Generics
2 {   public static void main(String[] args)
3     {   Example e = new Example("Hallo");
4         System.out.println(e.get());
5         e.set("Test");
6         System.out.println(e.get());
7     }
8 }
```

```
1 public class Test2 // Using a generic class
2 {   public static void main(String[] args)
3     {   Example<String> e = new Example<String>("Hallo");
4         System.out.println(e.get());
5         e.set("Test");
6         System.out.println(e.get());
7     }
8 }
```

Instantiation

Test2, Line 3: "`String`" is inserted as the actual type for "`T`".

Outline

Let's have an easy Assignment

1. Simplify your Highscore using a `LinkedList`.
   Shorter code is better code!

Let's have an easy Assignment

1. Simplify your Highscore using a `LinkedList`.
   Shorter code is better code!
2. Calculate the complexity of two different ways to insert a new score. No programming needed for this question.

Let's have an easy Assignment

1. Simplify your Highscore using a `LinkedList`.
   Shorter code is better code!
2. Calculate the complexity of two different ways to insert a new score. No programming needed for this question.
3. For those who like and enjoy:

Let's have an easy Assignment

1. Simplify your Highscore using a `LinkedList`.
   Shorter code is better code!
2. Calculate the complexity of two different ways to insert a new score. No programming needed for this question.
3. For those who like and enjoy:
   - Make your `LinkedList` a **generic**.

Let's have an easy Assignment

1. Simplify your Highscore using a `LinkedList`.
   Shorter code is better code!
2. Calculate the complexity of two different ways to insert a new score. No programming needed for this question.
3. For those who like and enjoy:
   - Make your `LinkedList` a **generic**.
   - Make your `GrowingArray` a **generic** *(tricky, non-standard)*.

Outline

Fibonacci in Java

Implementation

The aim is to put the following formula into code such that f_3 and the following 24 Fibonacci numbers are printed.

$$f_1 = f_2 = 1 \tag{1}$$

$$f_n = f_{n-1} + f_{n-2} \tag{2}$$

Fibonacci Code

```
1  public class PrintFibonacci
2  {    public static void main(String[] args)
3       {    int x, y, z;
4            x = 1; // start condition
5            y = 1; // start condition
6            for (int i=0; i<25; ++i)
7            {    z = x+y; // calculation
8                 System.out.println(z);
9                 x = y; // move the numbers
10                y = z;
11           }
12      }
13 }
```

Who understands this Code?

```
1  public class PrintFibonacci
2  {    public static int addPrint(int a, int b, int n)
3       {    int c = a + b;
4            System.out.println(c);
5            if (n > 1)
6            {    addPrint(b, c, n-1);
7            }
8            return c;
9       }
10
11      public static void main(String[] args)
12      {    addPrint(1, 1, 25);
13      }
14 }
```

Recursive Implementation

```
1  public class PrintFibonacci
2  {   public static int addPrint(int a, int b, int n)
3      {   int c = a + b;
4          System.out.println(c);
5          if (n > 1)
6          {   addPrint(b, c, n-1);
7          }
8          return c;
9      }
10     public static void main(String[] args)
11     {   addPrint(1, 1, 25);
12     }
13 }
```

Recursive Fibonacci Implementation

You can achieve the same goals using **recursive** and **iterative** programming. Often recursive code is smaller but a “bit more difficult to understand”.

Biggest Integer-Fibonacci-Number

Highest Fibonacci Number below MAX_VALUE

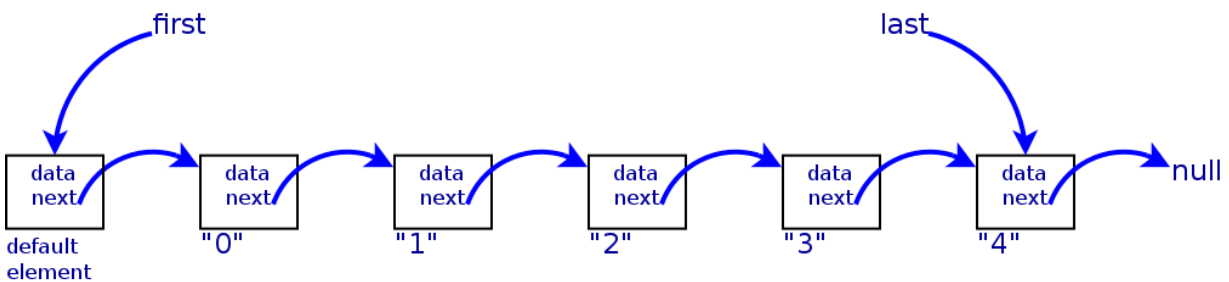
It's 2147483647, but this can't be calculated using the recursive solution (stack-overflow). Recursive solutions make the stack **exceed its limits** because of the repeated method invocations.

```
1  public class PrintFibonacci
2  {   public static void main(String[] args)
3      {   int x=1, y=1;
4          long z=1;
5          while (z < Integer.MAX_VALUE)
6          {   z = x+y;
7              x = y; // move the numbers
8              y = z;
9          }
10         System.out.println(z);
11     }
12 }
```

Outline

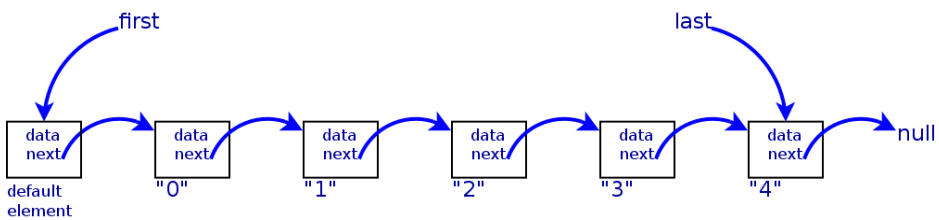
Example Solution: `insert`

```
1  public void insert(int pos, int dat)
2  {    ListElement le = new ListElement(dat);
3       ListElement bef = locate(pos-1);
4       ListElement aft = bef.next;
5       bef.next = le;
6       le.next  = aft;
7       if (aft == null)
8       {    last = le;
9       }
10 }
```



Example Solution: `delete`

```
1  public void delete(int pos)
2  {    if (pos < length())
3       {    ListElement bef = locate(pos - 1);
4            ListElement del = bef.next;
5            ListElement aft = del.next;
6            bef.next = aft;
7            if(last == del)
8            {    last = bef;
9            }
10      }
11 }
```



Example Solution: `sort`

```
1  private void sort()
2  {    boolean done = false;
3       while (!done)
4       {    done = true;
5            ListElement le = first.next;
6            while (le.next != null)
7            {    if (le.data < le.next.data)
8                 {    int tmp = le.data; // swap values
9                      le.data = le.next.data;
10                     le.next.data = tmp;
11                     done = false;
12                }
13                le = le.next;
14 }    }    } // saving space on the slide :-)
```

Questions from your side?

Please

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

All the Best!