Informatik II (D-ITET)

Tutorial 6

TA: Marian George, E-mail: marian.george@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Exercise Sheet 5: Solutions and Remarks

Variables & Methods
- `beginWithLowerCase`,
- `areVeryDescriptiveAnd`
- `upperCaseSeparated`
- `aMethodWhichHasAVeryLongName()`
- Classes with capital letters: class `MyClass`{ ... }`}
- C++ notation: attributes start with `m` (mValue, mNext), not the case with temporary and passed parameters (int tmp)

Please, comment more!
Solution Ex5.Q1, Q2, Q3 - A Simple Linked List

Solution Ex5.Q4 – List with Stack

Eclipse DEMO
Java Inheritance and Interfaces

Outline

- Person class example
- Inheritance
- Type compatibility
- Polymorphism
- `instanceof`
- Visibility rules
- Constructor and `super()`
- Final methods and class
- Abstract classes and methods
- Interfaces
- Abstract classes vs. interfaces
Class Person

Attributes

Accessors

Mutators

Person

Attributes

Name
Age
Address
PhoneNumber

toString

getAddress

getAge

getName

getAddress

getPhoneNumber

setAddress(newAddress)

setPhoneNumber(newPhoneNumber)

Getter and Setter Methods
public class Person {
    private String name;
    private int age;
    private String address;
    private String phone;

    public Person(String name, int age, String address, String phone) {
        this.name = name; this.age = age;
        this.address = address; this.phone = phone;
    }

    public String toString() {
        return getName() + " is " + getAge() +
            "old and lives in " + getAddress();
    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public String getAddress() { return address; }
    public String getPhoneNumber() { return phone; }

    .....
Class Student

**Person**
Name
Age
Address
PhoneNumber

toString()
getName()
getAge()
getAddress()
getPhoneNumber()

setAddress(newAddress)
setPhoneNumber(newPhoneNumber)

**Student**
Name
Age
Address
PhoneNumber

**Legi**

toString()
getName()
getAge()
getAddress()
getPhoneNumber()

getLegi()

setAddress(newAddress)
setPhoneNumber(newPhoneNumber)
public class Student extends Person {

private String legi;

public Student(String name, int age, String address, String phone, String legi) {
    super(name, age, address, phone);
    this.legi = legi;
}

public String toString() {
    return getName() + " is " + getAge() + "old, lives in " + getAddress() + " and has legi-nr.: " + getLegi();
}

public String getLegi() {
    return legi;
}
}
public class Student extends Person {

    private String m_legi;

    public Student(String name, int age,
                    String address, String phone, String legi){
        super(name, age, address, phone);
        m_legi = legi;
    }

    public String toString() {
        return getName() + " is " + getAge() + " old, lives in " +
            getAddress() + " and has legi-nr.: " + getLegi();
    }

    public String getLegi() { return m_legi; }
}
Inheritance

- Student extends Person
- Student can:
  - Add new fields
    - legi
  - Add new methods
    - getLegi()
  - Override existing methods
    - toString()
- Student cannot:
  - Remove fields
  - Remove methods
Why inheritance?

- Better design
- Code reuse
- Code «maintenance»
- Abstraction of the real world
Inheritance

Everything is an object!

by Sinipull for codecall.net
Upcasting

```
Cat c = new Cat();
System.out.println(c);
Mammal m = c; // upcasting
System.out.println(m);

/*
This printed:
Cat@a90653
Cat@a90653
*/
```

- Cat is still exactly the same Cat after upcasting, it didn't change to a Mammal, it's just being labeled Mammal right now. This is allowed, because Cat is a Mammal.

- Upcasting is done automatically, no need to do it manually
Upcasting

I'm still a Cat after upcasting, but compiler treats me as an Object

That means i can't do anything, that's specific for Animals... or Cats.

mammal

object

animal

cat

i can't!

Be a dog!

java.lang.ClassCastException: Cat cannot be cast to Dog

by Sinipull for codecall.net

Use static cast with upcasting to avoid overhead of dynamic casting
Downcasting

I'm still a Cat after upcasting, but compiler treats me as an Object

That means I can't do anything, that's specific for Animals... or Cats.

Downcasting must be done with dynamic cast to check if the object is actually of the derived class type

by Sinipull for codecall.net
Downcasting

Cat c1 = new Cat();

Animal a = c1; //automatic upcasting to Animal

Cat c2 = (Cat) a; //manual downcasting back to a Cat

• Downcasting must be done manually!
• Why? Multiple child classes
Downcasting

I can’t, because you don’t know if I’m a Cat, you must downcast me before I can do it.

by Sinipull for codecall.net
Static & Dynamic Cast

Person p = new Person(...);
Student s = new Student(...);
Employee e = new Employee(...);

Person ps = s \rightarrow ok
Person pe = e \rightarrow ok
Student sp = p \rightarrow compilation error
Student sps = ps \rightarrow compilation error
Student dps = (Student) ps \rightarrow ok (casting from base class to derived class)
Employee deps = (Employee) ps \rightarrow runtime error (ps points to object of class Student)
Static & Dynamic Cast

Person

Student

Employee

```
Person p = new Person(...);
Student s = new Student(...);
Employee e = new Employee(...);
```

p instanceof Person  →  true
p instanceof Student  →  false

s instanceof Person  →  true
s instanceof Student  →  true
The Object class in Java

- The Object class in Java
  - Is a superclass for all other classes defined in Java's class libraries, as well as for user-defined Java classes.
  - This does not include primitive types (char, int, float, etc.): they are not classes!

- When a class is defined in Java, the inheritance from the Object class is implicit, therefore:

  ```java
  public class MyClass {
  ......
  }
  ```

- is equivalent to:

  ```java
  public class MyClass extends Object {
  ......
  }
  ```
The Object class in Java

Quelle: sun.com
Visibility Rules

- **private members**
  - Private members in the base class are not accessible to the derived class, and also not to anyone else

- **protected members**
  - Protected members are visible to methods in a derived class and also methods in classes in the same package, but not to anyone outside

- **public members**
  - Everyone
Final Methods and Classes

- A derived class
  - Can accept the base class methods
  - Or can override the base class methods

- A method declared as final in the base class cannot be overridden by any derived class

- A final class cannot be extended!
  - E.g. Integer, Character,...
Abstract Classes

- Abstract method
  - Is a method that all derived classes **must** implement

- Abstract class
  - A class that has at least one abstract method

- If a class derived from an abstract class fails to override an abstract method, the compiler will detect an error
  - Eclipse: 'Hint-Bulb' provides help!
Interface

The interface in Java is the ultimate abstract class.

A class can implement many interfaces.

A class implements an interface if it provides definitions for all the methods „declared“ in the interface.

So, both abstract classes and interface provide a specification of what subclasses must do.

But....
Abstract Class vs. Interface
## Abstract Class vs. Interface

**Abstract class**
- An abstract class can provide complete code, default code, and/or just stubs that have to be overridden
- May declare methods as protected abstract
- A class may extend **only one** abstract class

**Interface**
- An interface cannot provide any code, much less default code
- All methods declared are implicitly public abstract
- A class may implement several interfaces

Example: Interface IStack

```java
public interface IStack {
    int size();
    void push(Object obj);
    Object pop();
    Object peek();
    boolean empty();
}
```

```java
public class MyStack implements IStack {
    private int size;

    public int size() {
        return size;
    }

    public void push(Object obj) {
        ...
    }
}
```
Example: Abstract Class BaseStack

```java
public abstract class BaseStack implements IStack {
    public abstract int size();
    public abstract void push(Object obj);
    public abstract Object pop();
    public Object peek() {
        Object top = pop();
        push(top);
        return top;
    }
    public boolean empty() {
        return size() == 0;
    }
}
```

```java
public class MyStack extends BaseStack {
    private GenericList first;

    public Object peek() {
        return first.value;
    }

    ...
}
```
HINTS FOR U6
Tips for U6

- Q1: Classes, Interfaces
- Q2: Interfaces and Implementation
- Q3: Polymorphism
- Q4: Stack (again): Voluntary Exercise Submission
Hints

A keyword represents an interface

```java
<<interface>>
ISensor
aktivieren()
lesen()
```

Notation for the dependance of the instantiation of the interface. Thermal sensor instantiates the iSensor interface

```java
<<interface>>
ISensor
aktivieren()
lesen()
```

```java
Wärmesensor
aktivieren()
lesen()
```

Keyword | Property
--- | ---

Section with attributes (detailed representation)

```java
<<gui>>
Window {abstract}
+ size: Area = (100,100)
# visibility: Boolean = true
+display()
+hide()
```

Section with operations (detailed representation)

```java
Person
name: String
vorname: String
```

```
Privatkunde
kundenummer: Integer
```

http://de.wikipedia.org/wiki/Klassendiagramm
Hints Ex6.Q2 – Factory Method

A factory method builds an object which implements a certain interface, but the inner functionality of the object is hidden.

→ Programmer 1 implements different lists which implement the \textit{IList} interface.

→ Programmer 2 uses lists but doesn’t want to know about the functionality. When Programmer 1 writes a new implementation, Programmer2 has to rewrite all lines of \texttt{new ListA()} as \texttt{new ListB()}. 

→ Programmer 1 puts a factory method at disposal and Programmer 2 can always call for example \texttt{Factory.giveMeNewList()} and gets an object from the newest implementation of the \textit{IList} interface.
Factory Method

Get me a Car

Car Factory

Car V1 Factory

Car VII Factory

Car GenX Factory

Car Prototype

Car V1

Car VII

Car GenX

Gets a car of Car Prototype
Hints Ex6.Q3 – Generic Lists

- Exercise sheet 5
  - Elements of the list: Integers
    - `int`

- Exercise sheet 6
  - Elements of the list: generic objects
    - `Object`

- Build your own utility class: `ListUtils`
  - `implements IListUtils`: manage generic lists
  - Compare with the utility classes in Q1 and Q3 of Exercise Sheet 5
  - This time the utility class is instantiated (not static)
Hints Ex6.Q3 – Generic Lists

class List {
    int value;
    List next;

    public List(int v, List e){
        value = v;
        next = e;
    }
}

value
  v
next
  e
Hints Ex6.Q3 – Generic Lists

class GenericList {
    Object value;
    GenericList next;

    public GenericList(Object v, GenericList e){
        value = v;
        next = e;
    }
}

<<Interface>>

Comparable

boolean smallerThan(Comparable rhs);
Hints Ex6.Q3 – Generic Lists

ListUtilsFactory

<table>
<thead>
<tr>
<th>public static IListUtils create()</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

public static void main(String[ ] args) {
    IListUtils utils = ListUtilsFactory.create();
    GenericList list = null;

    // add an object to the list
    Triangle t = new Triangle(2, 3);
    utils.add(list, t); // add() first checks if list is null
}
Hints Ex6.Q3a and c – Generic Lists

- Methods are not static anymore!
- Ex5.Q1: `toString`, `add`, `size`
  - Can be easily passed
- Ex5.Q3: `sort`
  - Similar (same idea)
  - Minimal interfacing This way, it can work with generic objects
  - Interface Comparable

```java
public interface Comparable {
    boolean smallerThan(Comparable rhs);
}
```

- You can cast as Comparable without checking its type
Hints Ex6.Q3b

- Implement area() method for Rectangle and Triangle
- Trivial
- Will be used in your tests for your list!
  - First implement…
  - …then look for errors in GenericLists 😊
Hints for E6.Q4 : A Stack Again

- Non-trivial

- Self-test: Whoever can do it will have no problem during the exam.
  - Promise 😊

- Combines the efficiency of arrays to the effortless growth of lists

- Implements the IStack Interface
  - Can be used in u6a2.StackFactory.create()
Have Fun!