Overview

- Debriefing Exercise 5
- Briefing Exercise 6
U05 Some Hints

Variables & Methods

- `beginWithLowerCase`
- `areVeryDescriptiveAnd`
- `toUpperCaseSeparated`
- `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)
public static List add(List list, int value) {
    return new List(value, list);
}

public static int size(List list) {
    if (list == null) return 0;
    return size(list.next) + 1;
}

public static int sum(List list) {
    if (list == null) return 0;
    return list.value + sum(list.next);
}

public static List last(List list) {
    if (list == null)
        return null;
    if (list.next == null)
        return list;
    return last(list.next);
}

public static List sublist(List list, int index) throws IndexOutOfBoundsException {
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list;
    return sublist(list.next, index - 1);
}

public static int valueAt(List list, int index) throws IndexOutOfBoundsException {
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list.value;
    return valueAt(list.next, index - 1);
}

public static int index(List list, int value) throws NoSuchElementException {
    if (list == null) throw new NoSuchElementException();
    if (list.value == value) return 0;
    return 1 + index(list.next, value);
}
public static void append(List list, int value) throws IllegalArgumentException
{
    if (list == null) throw new IllegalArgumentException();
    u5a1.Lists.last(list).next = new List(value, null);
}

public static void concat(List head, List tail) throws IllegalArgumentException
{
    if (head == null) throw new IllegalArgumentException();
    u5a1.Lists.last(head).next = tail;
}

public static List remove(List list, int index) throws IndexOutOfBoundsException
{
    if (list == null || index < 0) throw new IndexOutOfBoundsException();
    if (index == 0) return list.next;
    list.next = remove(list.next, index-1);
    return list;
}
U05.03 Sorting a Linked List

```java
public static List insertSorted(List list, int value) {
    if (list == null) return new List(value, null);
    if (value < list.value) return new List(value, list);
    list.next = insertSorted(list.next, value);
    return list;
}

public static List sort(List list) {
    if (list == null) return null;
    return insertSorted(sort(list.next), list.value);
}
```
Object Oriented Programming
Class Person

Person
Name
Age
Address
PhoneNumber

toString
getName
getAge
getAddress
getPhoneNumber

setAddress(newAddress)
setPhoneNumber(newPhoneNumber)

Attributes
Accessors
Mutators

Getter and Setter Methods
How do we implement class “Person”?

```java
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; }
}
```
What about students?

<table>
<thead>
<tr>
<th>Student</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Age</td>
<td>Age</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>PhoneNumber</td>
<td>PhoneNumber</td>
</tr>
</tbody>
</table>

`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; }
}
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!

Object

Animal
(Silently extends Object)

Mammal
(extends Animal)

Dog
(extends Mammal)

Cat
(extends Mammal)

upcasting

downcasting
Upcasting

```java
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
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.

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

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 Casting

Person

Student

Employee

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

Person ps = s ok
Person pe = e ok
Student sp = p Compile error
Student sps = ps Compile error
Student dps = (Student) ps Ok (casting from base class to derived class)
Employee deps = (Employee) ps Runtime error (ps points to object of class Student)
Static & Dynamic Casting

```
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
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 {
      .......
  }
  ```
Object class in Java

```
Object

Person

Student

Employee

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 provides help!
Interfaces

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
  - Not true since Java 8 (default code)
- All methods declared are implicitly public abstract
- A class may implement several interfaces

Example interface

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

public class MyStack implements IStack {
    private int size;

    public int size() {
        return size;
    }

    public void push(Object obj) {
        ...
    }
    ...
}
```
Example Abstract class

```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;
    }

    ...
}
```
Overview

- Debriefing Exercise 5
- Briefing Exercise 6
U06

- Q1: Classes, Interfaces and Casts
- Q2: Interfaces and their implementation
- Q3: Polymorphism
- Q4: ChunkedStack
Hints

A keyword represents an interface

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

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

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

Wärmesensor
aktivieren()
lesen()

Keyword
Property

Section with attributes (detailed representation)

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

Section with operations (detailed representation)

http://de.wikipedia.org/wiki/Klassendiagramm

Private client as a specialized Person

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

```
Privatkunde
kundennummer: Integer
```
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 `IList` interface.

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

→ Programmer 1 puts a factory method at disposal and Programmer 2 can always call for example `Factory.giveMeNewList()` and gets an object from the newest implementation of the `IList` interface.
U06.A03 a and c

- Methods are not static anymore!
  - Ex5.Q1: toString, add, size
  - Can be easily passed

- Interface Comparable

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

- You can cast as Comparable without checking its type
U06.A04 Stacks again!

- Not trivial – “advanced”
- Self-test: Whoever can do it will have no problem during the exam.
- Combines the efficiency of arrays to the effortless growth of lists
- Implement according to the interface
- Performance analysis
Have Fun!