

Voortgezet Programmeren

Lecture 5: Inheritance

Tommi Tervonen

Econometric Institute, Erasmus University Rotterdam

```
public class Student implements Keyable {  
    private String name;  
    private int nr;  
    public int getKey() {  
        return name.length();  
    }  
}  
  
public class Teacher implements Keyable {  
    private String name;  
    private String position;  
    public int getKey() {  
        return name.length();  
    }  
}
```

- A class can extend another one and add/redefine functionality
- The inheriting class is called the **subclass** and the inherited **superclass**

```
public class Person implements Keyable {  
    private String name;  
    public Person(String name) {  
        this.name = name;  
    }  
    public int getKey() {  
        return name.length();  
    }  
    public String getName() {  
        return name;  
    }  
}
```

```
public class Student extends Person {  
    private int nr;  
    ...  
    public int getNr() {  
        return nr;  
    }  
}
```

```
public class Teacher extends Person {  
    private String position;  
  
    ...  
  
    public String getPosition() {  
        return position;  
    }  
}
```

Subclass construction

```
public class Person {
    ...
    public Person(String name) { this.name = name; }
}
public class Student extends Person {
    public Student(String name, int nr) {
        super(name);
        this.nr = nr;
    }
}
```

- Construction starts by calling the most highest level class (Object) constructor, then its subclass, etc
- You can define the constructor of the superclass to call with `super()`, but this has to be the first line in the constructor
- If no `super` is called explicitly, the default constructor `super()` is executed

```
public class NumberedPerson extends Person {  
    private int nr;  
    ...  
    public String getName() {  
        return Integer.toString(nr) + ":"  
        + super.getName();  
    }  
}
```

- `this` = the current object
- `super` = the extended class
- You cannot refer higher than 1 level in the inheritance hierarchy
- You can widen the access specification (e.g. `private` to `public`) but not the other way around

Polymorphism and inheritance

```
Person p = new Student("tommi");  
Student s = (Student) p; // ok, as p is a Student  
Keyable k = s; // ok
```

- Polymorphism works with inheritance just like it does with interfaces
- In addition, with inheritance you can re-define functionality used in superclasses

Dynamic binding: example

```
public class Sum {  
    protected double a;  
    public Sum(double a) {  
        this.a = a;  
    }  
    public double getSum(double b) {  
        return getVal() + b;  
    }  
    protected double getVal() { return a; }  
}  
public class SquareSum extends Sum {  
    public SquareSum(double a)  
    { super(a); }  
    protected double getVal() {  
        return a * a;  
    }  
}
```

Inheritance interface and protected

- Subclass inherits all `public` and `protected` methods and fields
- `protected` access specifier defines the fields/methods to be accessible from sub-classes
- **Inheritance interface:** `public` and `protected` methods and fields

Abstract classes

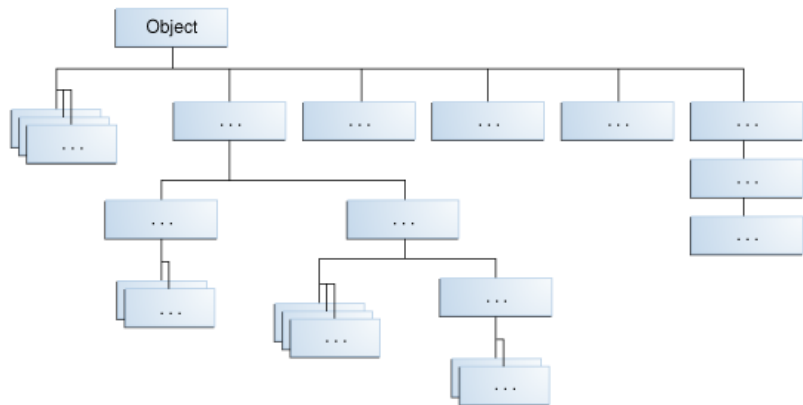
```
public abstract class MCSampler {  
    protected int misses;  
    ...  
    // public interface sampling method  
    public Sample sample() {  
        misses = 0;  
        return doSample();  
    }  
    // actual sampling method: inheritance interface  
    protected abstract Sample doSample();  
    // hit function  
    protected boolean isHit(Sample s) { ... }  
}
```

- Can leave some of the implementation to be defined in the subclasses
- Cannot be instantiated (like interfaces)

```
public class RejectionSampler extends MCSampler {  
    ...  
    protected Sample doSample() {  
        while (true) {  
            Sample s = drawSample();  
            if (isHit(s)) {  
                return s;  
            } else {  
                misses++;  
            }  
        }  
    }  
    private Sample drawSample() { ... }  
    ...  
}
```

```
public class MCMCSampler extends MCSampler {  
    private Sample curSample;  
    ...  
    protected Sample doSample() {  
        Sample newSample = drawMCMCSample(prevSample);  
        if (isHit(newSample)) {  
            curSample = newSample;  
        } else {  
            misses++;  
        }  
        return curSample;  
    }  
    private Sample drawMCMCSample(Sample s) { ... }  
    ...  
}
```

Inheritance hierarchies



(Super)[∞] class Object

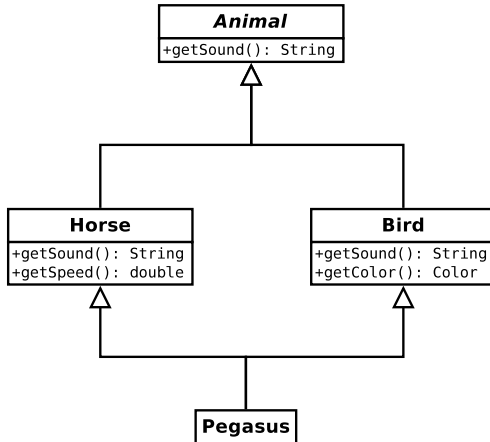
- All objects inherit Object implicitly unless they inherit another class
- Object defines a few useful methods you can override that are used widely in the standard library:
 - `public String toString()` - for obtaining the object's String representation
 - `protected clone()` - makes a shallow copy of the object. Can be overridden to provide an implementation. DO NOT USE.
 - `public boolean equals(Object other)` - returns true if this object is equal to other, false otherwise (implementation in Object is of reference equality)
 - `public int hashCode()` - gives hash value of the object, used in e.g. hash maps of the standard library (next lecture)

Inheritance in exceptions

- Unchecked exceptions extend `RuntimeException` or `Error`
- Checked exceptions extend `Exception` (but not `RuntimeException`)
- Thus your own checked exceptions should extend `Exception`:

```
public class MyException extends Exception {  
    public MyException(String reason) {  
        super(reason);  
    }  
    public MyException() {  
        // super() called implicitly  
    }  
}
```


Multiple inheritance



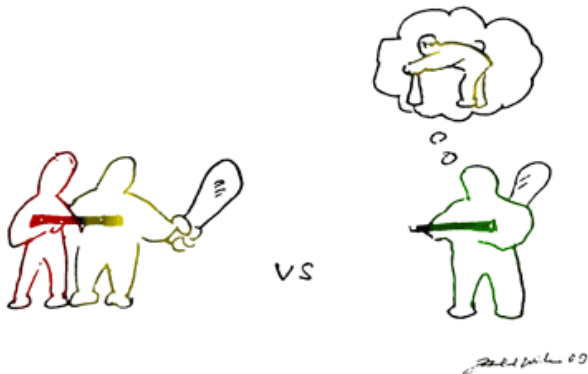
- Not allowed in Java
- Use interfaces instead - you can extend only 1 class but implement multiple interfaces

```
public interface Collection {  
    public void add(Object o);  
    public void remove(Object o);  
}
```

```
public interface List extends Collection {  
    public void add(int index, Object o);  
    public void remove(int index);  
}
```

- Simply adds methods to be implemented (+ type information)

Inheritance vs composition



- You can in many cases choose between inheritance (is-a) and composition (has-a)
- Inheritance is not a silver bullet

Next exercise

