Inheritance. Overwriting and Overloading

Prof. Dr. Dr.-Ing. J. Ovtcharova

Dipl. Wi.-Ing. Dan Gutu

9th of January 2009
Outline of Exercise 6

Content of Exercise 6

1. Inheritance

2. Overwriting and Overloading
Inheritance: Overview

Deriving classes from other classes is called **inheritance**

- The base class is called **superclass** or **upper class** or **parent class**.

- The derived class is called **subclass** or also **child class**.

- The subclass
  - inherits the variables and methods of the superclass
  - may have additional variables and methods.

- A class has **at most** one superclass,

  Java **does not provide multiple inheritance**
  (inheritance from several classes)!

- But: a class may have several subclasses!

[Diagram of UML-Notation showing a triangle connecting Circle to GraphicCircle]
public class GraphicCircle extends Circle {

    public Color outline, fill; // new variable

    public void draw(DrawWindow dw) { // new method
        dw.draw(x,y,r,outline,fill);
    }
    // The class GraphicCircle inherits automatically all variables

    } // and methods of the class Circle
### Usage

```java
public class GraphicCircle {
    private Circle c; // declares object c of type Circle

    private Color outline, fill; // more instance variables

    // 2 methods as in class Circle:
    public double circumference() {
        return c.circumference();
    }

    public double area() {
        return c.area();
    }

    // one more instance method:
    public void draw(DrawWindow dw) { /* Code missing */ }
}
```

### Inheritance

```java
public class GraphicCircle extends Circle {
    // Here we need only the new variables and methods - because of the inheritance.
    private Color outline, fill;

    // these methods and variables describe what is only specific for the subclass.
    public void draw(DrawWindow dw) {
        dw.draw(x,y,r,outline,fill); }
}
```
A Class Hierarchy Diagram

The class called **Object** is the root of the hierarchy system (in Java)

-> Has no class before
-> The methods of each object can be used by every class
Calling Constructors of the Superclass

```java
public class GraphicCircle extends Circle {
    public Color outline, fill;

    // Constructors
    public GraphicCircle(double x, double y, double r,
            Color outline, Color fill) {
        this.x = x;
        this.y = y;
        this.r = r;
        this.outline = outline;
        this.fill = fill;
    }

    // Methods ...
}
```

**this**
- "this" object
  - Calling methods and accessing instance variables of the instance of "this" class

**super**
- "superordinate" object
  - Calling methods and accessing instance variables of the instance of the super-class

```java
public class GraphicCircle extends Circle {
    public Color outline, fill;

    // Constructors
    public GraphicCircle(double x, double y, double r,
            Color outline, Color fill) {
        super(x, y, r);
        this.outline = outline;
        this.fill = fill;
    }

    // Methods ...
}
```
Hidden variables from inheritance

- An instance variable from a superclass is **hidden**, if in the subclass an instance variable with identical name is declared or defined.

- An instance of the subclass contains both the instance variables from the **superclass** and from the **subclass**.

- A **hidden instance variable** is not directly visible in the instance of the subclass.

- Analogously, in case that the subclass contains a class variable with an identical name to a class variable from the superclass, this variable will be hidden in the subclass.
Access on instance variables from an instance of class C:

- `this.x`
- `super.x`
- `((B)this).x`
- `((A)this).x`
- `super.super.x`

Illegal access on instance variables from an instance of class C.
Hidden Instance Variables with Inheritance (2)

ClassA

x

ClassB

x

ClassC

Access on instance variables from an instance of class C:

x

this.x

super.x

((ClassB)this).x

((ClassA)this).x

super.super.x

illegal

Notice:
If no variable x is defined in ClassC, this.x accesses the variable defined in the upper class!
Hidden Class Variables with Inheritance (1)

Access on class variables from an instance of class C:

- **Illegal Access** on class variables from an instance of class C.

- **Casting**
  - `x`
  - `(B)x`
  - `(A)x`
  - `super.super.x`

- **Illegal**
Hidden Class Variables with Inheritance (2)

ClassA
(static) x

ClassB
(static) x

ClassC

Access on class variables from an instance of class C:

x
super.x
(ClassB)x
(ClassA)x
super.super.x

illegal
Outline of Exercise 6

Content of Exercise 6

1. Inheritance
2. Overwriting and Overloading
Overwriting and overloading methods

- **overwriting**: re-defining a method in a sub-class
  - A method of the upper class will be overwritten in the sub-class in which a method with the same name and signature (parameter list) will be defined.
  - The correct way to link a method will then be performed (dynamic linking), depending on the object type of the instance (from the upper class or the sub-class) through which the method will be called.
  - Methods defined to be **final** in the superclass cannot be overwritten in the subclasses and will be linked statically (during compilation)

- **overloading**: the addition of variables to a methods parameter list (within a class or sub-class)
  - At least 2 methods exist with the same name but with different signatures (parameter lists) within a class or the hierarchy of the class
Method Overwriting (1)

```
public class Circle {
    // Attributes and Constructors
    // ...

    // Methods
    public double area() {return 3.14159 * radius * radius;}
    // ...
}

public class Ellipse extends Circle {
    // Attributes and Constructors
    // ...

    // Methods
    public double area() {return 3.14159 * a * b;}
    // ...
}
```
class A {
    int i = 1;
    int f() { return i; }
}

class B extends A {
    int i = 2;
    int f() { return -i; }
}

B b = new B();
System.out.println(b.i);  // refers to B.i: 2
System.out.println(b.f()); // refers to B.f(): -2

A a = (A)b;  // transforms b into an A instance
System.out.println(a.i); // refers to A.i: 1
System.out.println(a.f()); // refers to B.f(): -2

Diagram:

- New object of type B
- B.b = new B();
- B.f() = 2
- A.a = (A)b;
- A.i = 1
- A.f() = -2

Diagram nodes:
- B.i = 2
- A.i = 1
- B.f() = -2
- A.f() = -2

Diagram edges:
- b → B.i
- b → B.f()
- a → A.i
- a → A.f()
Hidden instance methods with Inheritance (1)

Access on instance methods from an instance of class C:

- `f()`
- `this.f()`
- `super.f()`
- `((B)this).f()`
- `((A)this).f()`
- `super.super.f()`

Illegal
Access on instance methods from an instance of class C:

- \( f() \)
- \( \text{this}.f() \)
- \( \text{super}.f() \)
- \( ((\text{ClassB})\text{this}).f() \)
- \( ((\text{ClassA})\text{this}).f() \)
- \( \text{super}.\text{super}.f() \)

Notice:
If no method \( f() \) is defined in ClassC, \( \text{this}.f() \) accesses the method defined in the upper class!

Illegal
Access on class methods from an instance of class C:

- \texttt{(static) f()}
- \texttt{super.f()}
- \texttt{(B)f()}
- \texttt{(A)f()}
- \texttt{super.super.f()}

\textbf{illegal}

\textbf{Casting}
Hidden class methods with Inheritance (2)

Access on class methods from an instance of class C:

ClassA

(static) f()

ClassB

(static) f()

ClassC

f()

(super.f())

((ClassB)f())

((ClassA)f())

super.super.f()

illegal
public class A {
    protected int i = 1;
    public int f() {
        return i;
    }
}

public class B extends A {
    private int i = 2;
    public int f() {
        return -i;
    }
}

A a = new A();
System.out.println(a.i); System.out.println(a.f());
B b = new B();
System.out.println(b.i); System.out.println(b.f());
A aa = (A)b;
System.out.println(aa.i); System.out.println(aa.f());

new object of type A
refers to A.i: 1
refers to A.f(): 1
new object of type B
refers to B.i: 2
refers to B.f(): -2
transforms b into an A
instance refers to A.i: 1
Refers to B.f(): -2
class A {
    int i = 1;
    int f() { return i; }
}

class B extends A {
    int i;  // this variable hides i in A
    int f() {  // this method overwrites f() in A
        i = super.i + 1;  // A.i is addressed like this
        return super.f() + i;  // A.f() is called like this
    }
}

Calling Hidden Methods (2)
public class ExpensiveCarpet {
    private String location = " Japan ";
    public void productionInfo() { System.out.println(" manufactured in " + location); }
}

public class Carpet extends ExpensiveCarpet {
    protected String location = " China ";
    public void productionInfo() { System.out.println(" produced in " + location); }
}

Carpet oneCarpet = new Carpet();
System.out.println(oneCarpet.location);
oneCarpet.productionInfo();

ExpendiveCarpet anExpensiveCarpet = new ExpensiveCarpet();
System.out.println(anExpensiveCarpet.location);
anExpensiveCarpet.productionInfo();

ExpensiveCarpet aCarpet = (ExpensiveCarpet) oneCarpet;
System.out.println(aCarpet.location);
aCarpet.productionInfo();
Method overloading

```
public class Circle {
    public double x, y, r;

    // Constructors and methods
    // ....

    public void setCenter(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public void setCenter(Point p) {
        c.x = p.getX();
        c.y = p.getY();
    }
}
```

```
Circle c1 = new Circle(0.0, 0.0, 1.0);
c1.setCenter(1.0, 3.0);
```

```
public class Point {
    public double x, y;

    // Constructors and methods
    // ....

    public double getX() { return x; }
    public double getY() { return y; }
}
```

```
Point p = new Point(1.0, 3.0);
c2.setCenter(p);
```