



Overloading and Overriding

s20



Overloaded Methods

- Let you reuse the same method name in a class
 - with different arguments
 - Optionaly with different return type



Overloaded Methods

- Can change the access modifier
- Can declare new or broader checked execptions
- Can be overloaded either in
 - The same class
 - In a subclass
- A constructor can be overloaded within the same class



Overloaded Methods

- Overloading: Example

```
class MyCalculator {  
    int sum(int x, int y) { return x+y;}  
    int sum(int x, int y, int z){ return x+y+z;}  
    float sum(float x, float y){ return x+y;}  
}  
  
public class Calculator {  
    public static void main(String args[]) {  
        MyCalculator c = new MyCalculator();  
        c.sum(2,3);  
        c.sum(2,3,4);  
        c.sum(2.0f,3.0f);  
    }  
}
```

s18



Overloaded Mehtods

- Changing only the return type is not allowed

```
public class Foo {  
    void go() {}  
    String go() { //return null  
        return null;  
    }  
}
```



Overriden Methods

- A method of a sub class can be overridden if the following is preserved from the superclass
 - The same method name
 - The same arguments
 - The return type can be different (Java 5)
 - The acces modifier can be equal or more public



Overriden Methods

- A method cannot be overridden if in the suppperclass it is declared as
 - final
 - abstract
 - static



- Example:

```
public class Animal {  
    public void printYourself() {  
        System.out.println("I' m an Animal" );  
    }  
}  
  
class Horse extends Animal {  
    public void printYourself() {  
        System.out.println("I' m a Horse");  
    }  
}
```

Overriden Methods

- Since Java 5 it is allowed to change the return type if
 - The new return type is a subtype of the overridden method's return type

```
Class Alpha {  
    Alpha doStuff(char c) {  
        return new Alpha();  
    }  
}  
Class Beta extends Alpha {  
    Beta doStuff(char c) {  
        return new Beta();  
    }  
}
```

Overriden Methods

- Does this code compile?

```
public class Foo {  
    void go() {}  
}  
  
class Foo2 extends Foo {  
    String go() { //return null  
        return null;  
    }  
}
```

Overriden Methods

- Constructors cannot be overridden
 - One reason is constructors are not inherited

Legal return types

```
1. class Driver {  
2.     Driver shift() { return this; }  
3. }  
4. class BusDriver extends Driver {  
5.     // insert code here  
6. }
```

Which code inserted at line 5 will compile? (choose all that apply)

- a) Driver shift() { return this; }
- b) BusDriver shift() { return this; }
- c) Object shift() { return this; }
- d) int shift() { return 1; }
- e) int shift(int x) { return 1; }
- f) Object shift(int x) { return this; }

- Activity (overloading): implement the following interface

```
interface Imatrix {
    public void print(int v[]);
    public void print(int m[][]);
    public void multiply(int m[][], int v[], int r[]);
    public void multiply(int a[][], int b[][], int r[][]);
}
```

Polymorphism

s20

Polymorphism

- Ability of a single variable of a given type to
 - be used to reference objects of different types
 - to automatically call the method that is specific to the type of object the variable references

Call specific method

```
class Animal {
    String kind;
    Animal(String k) { kind = k; }
    void getKind() { System.out.println("I'm a " + kind); }
    void getSound() { System.out.println("undefined"); }
}
class Dog extends Animal {
    Dog(String k) { super(k); }
    void getKind() { System.out.println("I'm a " + kind); }
    void getSound() { System.out.println("woof woof"); }
}
class Cat extends Animal {
    Cat(String k) { super(k); }
    void getKind() { System.out.println("I'm a " + kind); }
    void getSound() { System.out.println("miauuuu"); }
}
class Duck extends Animal {
    Duck(String k) { super(k); }
    void getKind() { System.out.println("I'm a " + kind); }
    void getSound() { System.out.println("quack quack"); }
}
```

```
public class Polymorphism {
    public static void main(String args[]) {
        Animal[] animals = { new Dog("dog"),
                               new Cat("cat"),
                               new Duck("duck") };
        for(int i=0; i<animals.length; i++) {
            animals[i].getKind();
            animals[i].getSound();
        }
    }
}
```



Polymorphism

- Upcasting

- Casts to a super class. Example
 - `Employee employee = new Manager();`
- Implicit
- Only the resources of the super class are visible
- The behaviour is not of the superclass but of the subclass



```
class Employee {  
    public void work() {  
        System.out.println("I am an employee.");  
    }  
}  
  
class Manager extends Employee {  
    public void work() {  
        System.out.println("I am a manager.");  
    }  
    public void manage() {  
        System.out.println("Managing ...");  
    }  
}
```

```
public class Polymorphism2 {  
    public static void main(String[] args) {  
        Employee employee;  
        employee = new Manager(); //upcasting  
        employee.work();  
        //employee.manage(); //not visible  
    }  
}
```



- Quiz: compile the code to test the upcast



Polymorphism

- Downcast


- Casts to a subclass
- Only valid when the object was previously upcasted
 - Valid
 - `Employee employee = new Manager();`
 - `Manager manager = (Manager) employee;`
 - Not valid
 - `Employee employee = new Employee();`
 - `Manager manager = (Manager) employee;`




Polymorphism

- Downcast

- Requires explicit cast
- The resources of the subclass are visible
- The behaviour is of the subclass



```
class Employee {  
    public void work() {  
        System.out.println("I am an employee.");  
    }  
}  
  
class Manager extends Employee {  
    public void work() {  
        System.out.println("I am a manager.");  
    }  
    public void manage() {  
        System.out.println("Managing ...");  
    }  
}  
  
public class Polymorphism2 {  
    public static void main(String[] args) {  
        Employee employee;  
        employee = new Manager(); //upcasting  
        Manager manager = (Manager) employee; //downcasting  
        manager.work();  
        manager.manage(); //visible  
    }  
}
```

- 
- Quiz: compile the code to test the downcast