

*Core Java Course*  
*Inheritance and Polymorphism*  
(Chapter 5)  
Prof. Jeffrey Blessing  
MSOE

---

---


---

---

---

---

---



*Agenda*

- ? Homework/questions
- ❖ Inheritance
- ❖ Polymorphism
- ❖ Reflection package
- ❖ Proper usage of inheritance (design rules)

2

---

---


---

---

---

---

---



*Why is Inheritance Important*

- ❖ Encourages code re-use
- ❖ Allows “Core” of system to remain intact while developing “around the fringes”
- ❖ In Java many actions are impossible without using inheritance
- ❖ Appears to be very similar to C++ or other languages but there are essential differences

3

---

---

---

---

---

---

---

*Vocabulary*

- ❑ **Extends** - a new class that inherits from existing class
- ❑ **Superclass** - the existing class (also, base class or parent class)
- ❑ **Subclass** - the new class (also, derived class or child class)
- ❑ **Polymorphism** - object's ability to decide which method to use depending on its position in the inheritance hierarchy

4

---

---

---

---

---

---

---

*What is Inheritance?*

- ❖ Common data structures and behavior are implemented in the super class
- ❖ Subclasses only have to implement differences in the data in behavior
- ❖ Normally, subclasses carry additional features not available in the superclass
- ❖ The inheritance chain can be as long as we need

5

---

---

---

---

---

---

---

*Questions*

- ? Who is superior: superclass or subclass?
- ? Who has richer functionality: superclass or subclass?
- ? Can subclass variable be assigned to a superclass object?
- ? Can superclass variable be assigned to subclass object?

6

---

---

---

---

---

---

---

***Inheritance Example***

```

class Car extends Vehicle
{
    public void Car(Point pos, int nDoors)
    {
        super(pos);
        numDoors = nDoors;
    }
    public int GetNumDoors()
    {
        return numDoors;
    }
    private int numDoors;
}

```

7

---

---

---

---

---

---

---

---

***Inheritance Implementation***

- ❖ “Car extends Vehicle” means that a Car has all the data structures and methods that a Vehicle does
- ❖ Also a Car has something in addition to a Vehicle. In this particular case it’s a numDoors attribute and a method GetNumDoors() to access it

8

---

---

---

---

---

---

---

---

***Access to the superclass***

- ❖ In C++ use scope operator
  - SuperClassName::MethodName()
- ❖ In Java use super keyword
  - super(a, b, c) - the superclass constructor
  - super . - the superclass “pointer”  
(super.turnLeft());
- ❖ In C++ any super class in the whole hierarchy is accessible
- ❖ In Java only immediate superclass is accessible

9

---

---

---

---

---

---

---

---

### Subclass or Superclass?

- ❖ Use inheritance when your subclass can be always used as the superclass too

```
Car c = new Car();
Vehicle v = c; // a car is as good as vehicle
```

- ❖ The opposite isn't true

```
Vehicle v[] = new Vehicle[10];
Car c = (Car)v[1]; // this would result in a
// run-time error, since a Vehicle has
// fewer methods than a Car expects to
// be able to access
```

10

---

---

---

---

---

---

---

---

### Polymorphism

- ❖ Need at least one superclass and different subclasses
- ❖ The polymorphic method should be defined in the superclass
- ❖ Subclasses re-implement the method
- ❖ A superclass-type object array is used to reference a group of different subclass objects
- ❖ At run time the same method is invoked on all the array elements but actual objects behave differently

11

---

---

---

---

---

---

---

---

### Polymorphism (cont.)

- ❖ Compare to C++ Java methods are virtual by default
- ❖ Late or dynamic binding is used to invoke all (virtual) polymorphic methods
- ❖ Late binding means that decision is made at run-time and is based on the actual object type (not known at compile-time)
- ❖ Traditional methods use static binding. All the decisions are made at compile-time and based on the object variable type (known at compile-time)

12

---

---

---

---

---

---

---

---

### If We Don't Want Polymorphism

- ❖ Use keyword `final` to prevent methods from being virtual
- ❖ Lose flexibility
- ❖ Gain efficiency
  - don't have to decide at run-time
  - in-line methods are possible
  - optimization is possible
- ❖ Gain security and determinism
  - can make sure that class behavior cannot be altered

13

---

---

---

---

---

---

---

---

### Casting

- ❖ Used to convert from basic type to another
- ❖ Also, used to convert from class type to another
  - `Car myCar = (Car)vehiclesParked[7];`
- ❖ Cannot use casting when the source has lower capacity than the target. This will cause a run-time exception
- ❖ Don't have to cast when assigning a superclass variable to a subclass object
- ❖ Use `instanceof` operator to make sure you can cast
  - `if( v[7] instanceof Car) //...`

14

---

---

---

---

---

---

---

---

### Casting Java vs. C++

- ❖ C++: `static_cast<Type>` and `dynamic_cast<Type>`
- ❖ Run-time casting in Java similar to `dynamic_cast`
- ❖ Differences
  - if `dynamic_cast` fails in C++ it returns a null pointer
  - Java does not have pointers
  - if casting fails in Java it generates an exception
- ❖ To avoid run-time problems always do `instanceof` check before dynamic casting. In this case it becomes an equivalent of `dynamic_cast`

15

---

---

---

---

---

---

---

---

### Abstract Classes and Methods

- ❖ Abstract class - a concept class that cannot have instances
- ❖ Abstract method - a method that has no implementation in the given class (the class itself must be declared as `abstract`)
- ❖ Unlike C++ an abstract class in Java does not have to have abstract methods
- ❖ We can view an abstract class as a contract for the future implementation of certain methods or behavior in subclasses (compared to interfaces)

16

---

---

---

---

---

---

---

---

### Class Object (java.lang.Object)

- ❖ Class `Object` is used as a universal (Cosmic) superclass (compare to `CObject` in MFC)
- ❖ Any class in Java extends class `Object`
- ❖ Important methods
  - `Class getClass()`
  - `boolean equals(Object obj)`
  - `Object clone()`
  - `String toString()`

17

---

---

---

---

---

---

---

---

### Object wrappers

- ❖ Wrappers are used to simulate and extend behavior of basic data types
  - `Integer, Long, Double, ...`
  - `static String toString(int i, int b)`
  - `static int parseInt(String s)`
  - `static Integer valueOf(String s)`
- ❖ Wrapper classes are still immutable
- ❖ Need to be used in generic programming approach when basic data types should be treated in the same way as any class objects

18

---

---

---

---

---

---

---

---

**Run-time type Identification**

- ❖ During run-time type information can be obtained by calling `getClass()` method in the Object class
  - `Car c;`
  - `Class cl = c.getClass();`
- ❖ `getClass()` returns an instance of Class type
- ❖ Useful Class methods (`java.lang.Class`)
  - `String getName()`
  - `String getSuperclass()`
  - `Class[] getInterfaces()`
  - `String toString()`
  - `static Class forName(String clName)`
  - `Object newInstance()`

19

---

---

---

---

---

---

---

---

**Reflection (`java.lang.reflect`)**

- ❖ Reflective programming is a technique that allows a program to analyze the structure and capabilities of a class or a component at run-time
- ❖ Examples
  - Visual Basic/OLE Controls (VBX's and OCX's)
  - ActiveX controls
  - JavaBeans
- ❖ Reflection classes/features are added in Java 1.1
  - `Class`, `Field`, `Method`, `Constructor`

20

---

---

---

---

---

---

---

---

**Code Examples**

- ❖ Polymorphism
  - Manager vs. Employee (`ManagerTest.java`)
  - Mailbox (`MailboxTest.java`)
- ❖ Reflection
  - Class snapshot (`ReflectionTest.java`)
  - Generic Growing Array (`ArrayGrowTest.java`)

21

---

---

---

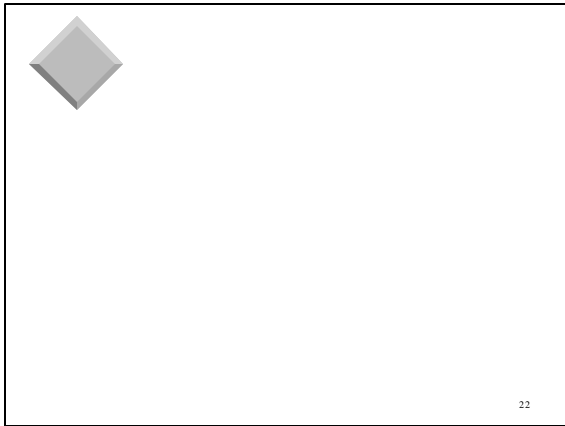
---

---

---

---

---



---

---

---

---

---

---

---