Java Programming Language - Basic

Peter.Cheng

founder_chen@yahoo.com.cn

http://www.huihoo.com

2004-04

Course Goal

- The main goal of this course is to provide you with the knowledge and skill necessary for object-oriented programming of java application. In this course, you will learn Java programming language syntax and object-oriented concepts, multithreading, and networking.

Course Overview

This course covers the following areas:

- OO Concept, CRC (Class, Responsibility, Collaboration)
- Syntax of the Java programming language
- Object-oriented concepts as they apply to the Java programming language
- Multithreading
- Networking

Course Map

The Java Programming Language Basics

- Object-Oriented Programming
- Identifiers, Keywords, and Types

More Object-Oriented Programming

- Inheritance
- Advanced Class Features

Advanced Java Programming

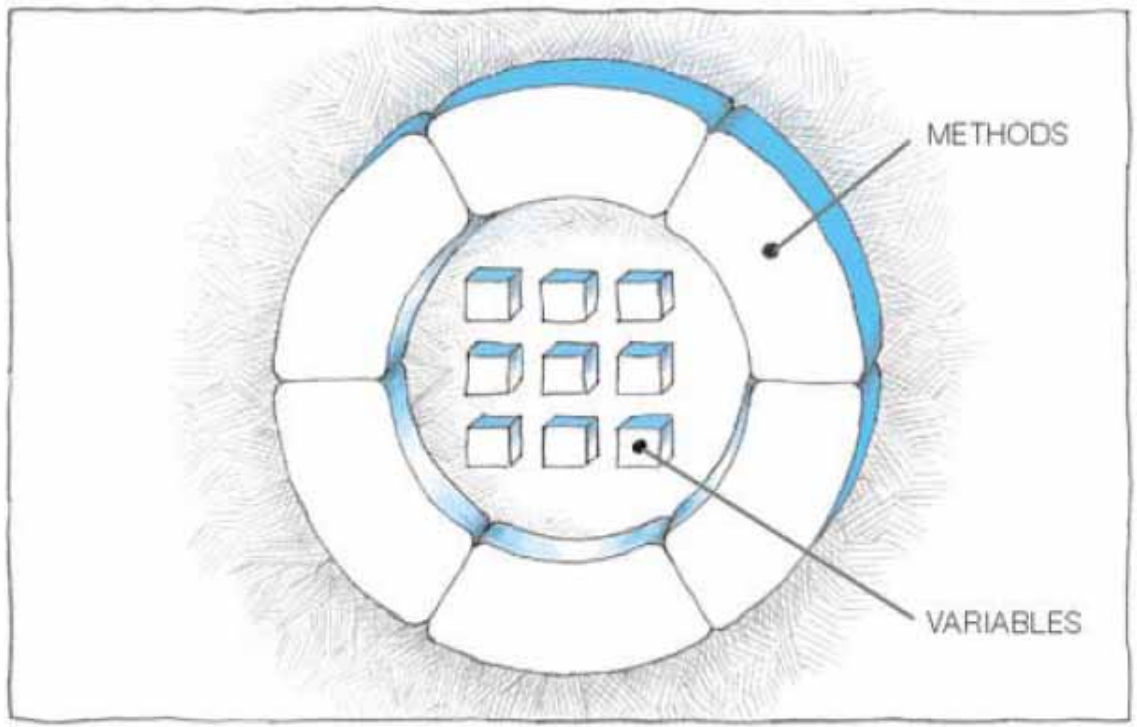
- Threads
- Networking

Object

- **What's an object?**

  A person, thing, concept, event, screen, or report. Objects both know things (that is , they have data) and they do things (that is, they have functionality)

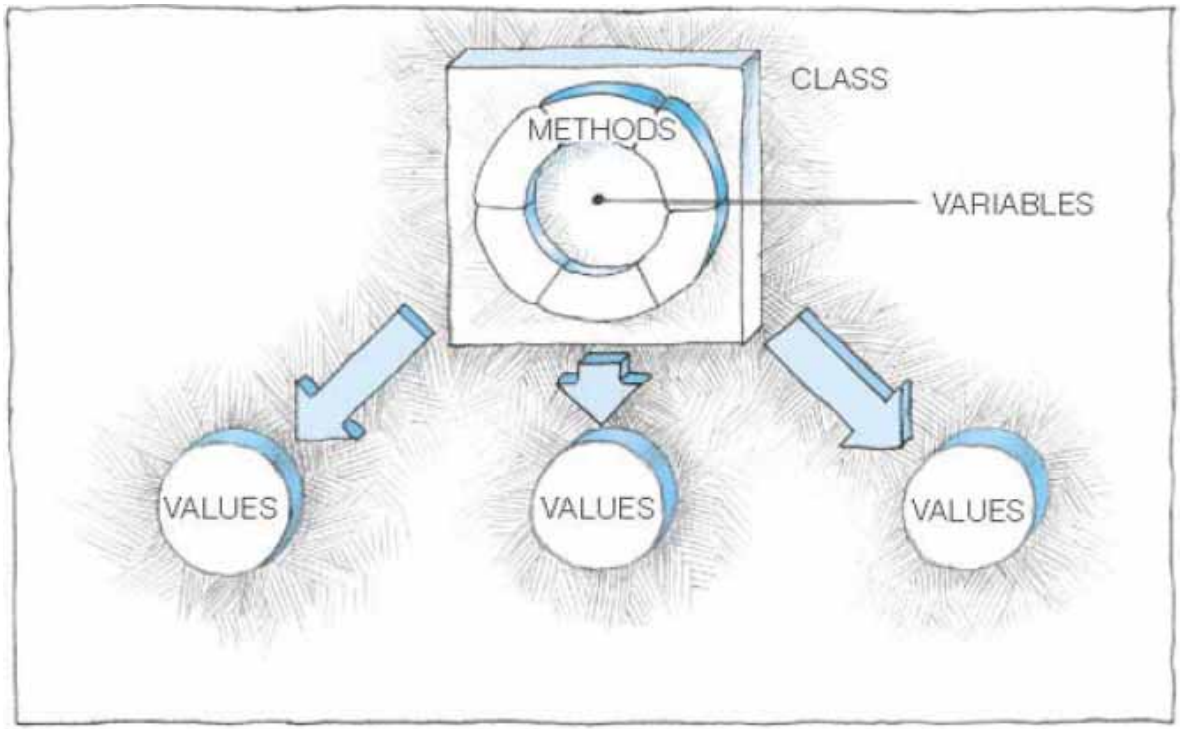
An object



Class

- The class is used to describe a set of like things. It describes all of these elements in a general way but allows each instance of the class to very in nonessential features.
- The class is abstract and conceptual, the instances are concrete, physical objects.

A class and it's instances



人 Class

- 下图中的“人”类，包含两个属性：姓名和年龄以及改变年龄和姓名的方法。

对象类

Person	← 类名
name age	← 属性
changeName() incrementAge()	← 方法

对象实例

P1:Person
Name=“John” Age=20

对象实例

P2:Person
Name=“Mary” Age=18

Responsibilities

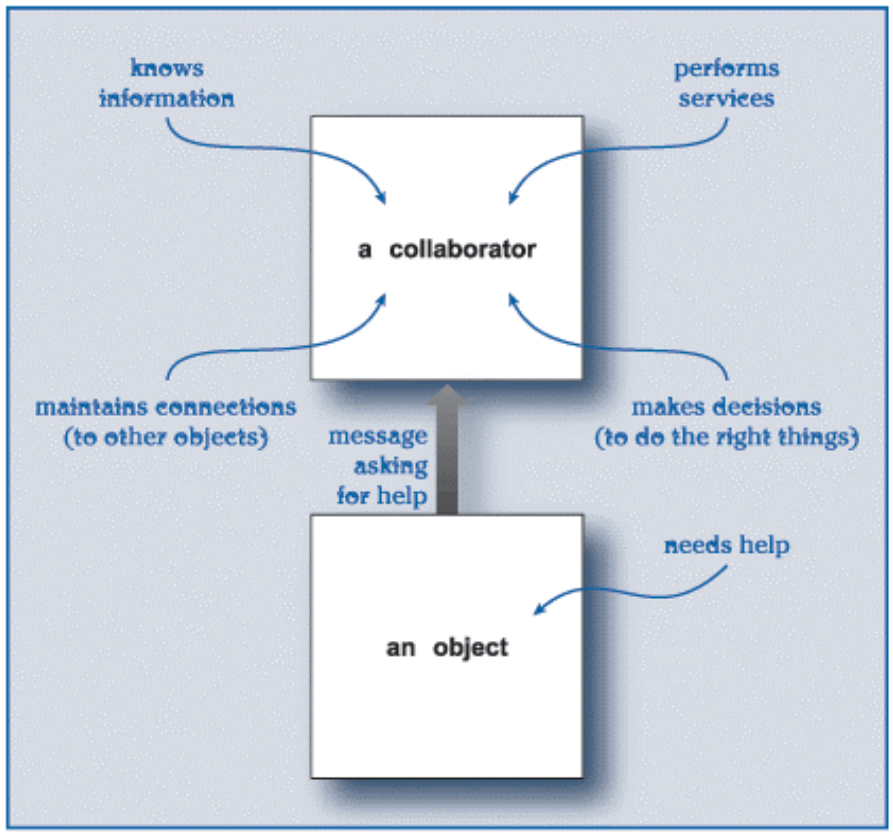
- **What are Responsibilities?**

  Responsibilities are general statements about software objects

- **Three major items:**
  - The actions an object performs
  - The knowledge an object maintains
  - Major decisions an object makes that affect others

Collaborations

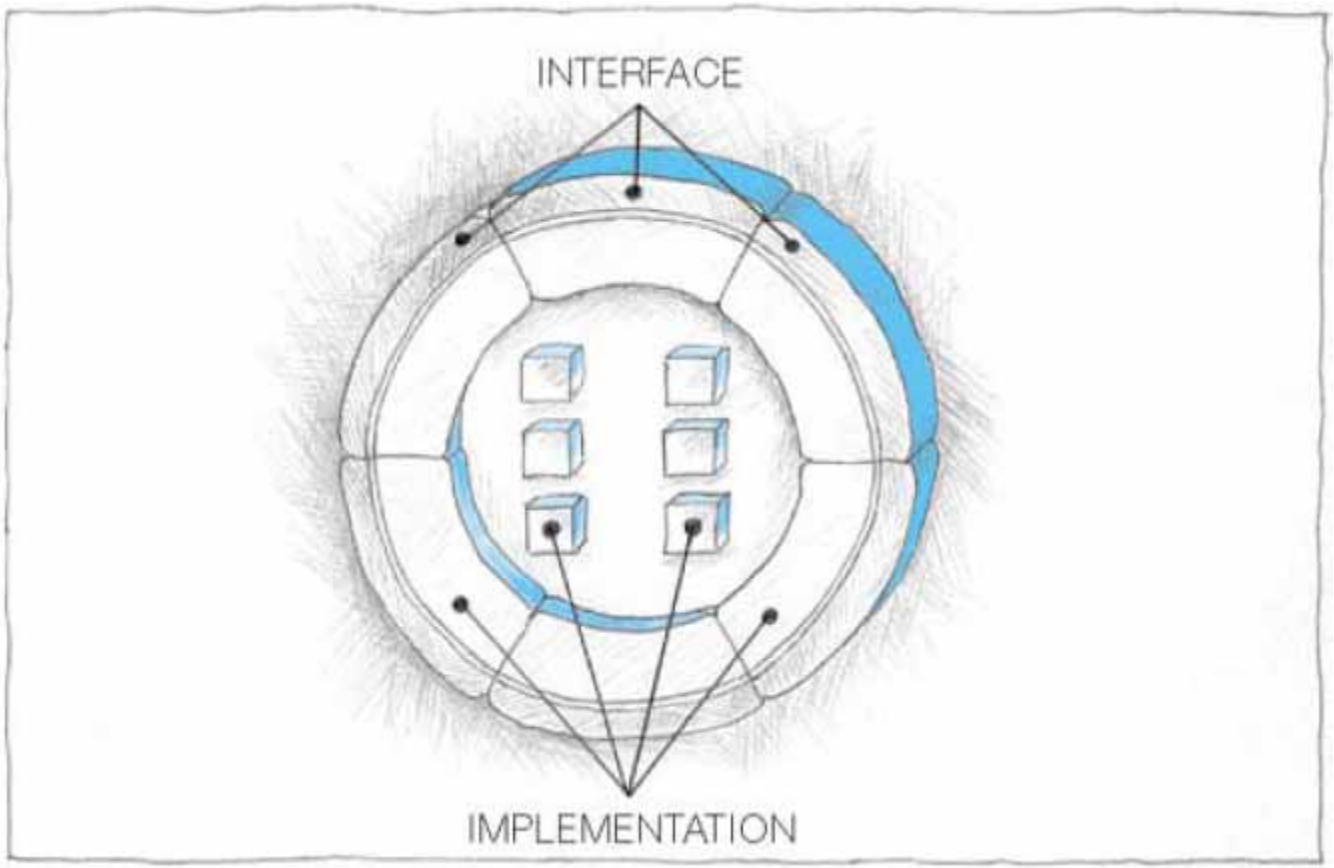
- Collaborations are requests from one object to another



The three keys of OO

- There *is* an industry-standard definition of object-oriented technology, and it can be summarized in terms of three key concepts:
  - Objects that provide 封装(encapsulation) of procedures and data
  - Messages that support 多态(polymorphism) across objects
  - Classes that implement 继承(inheritance) within class hierarchies

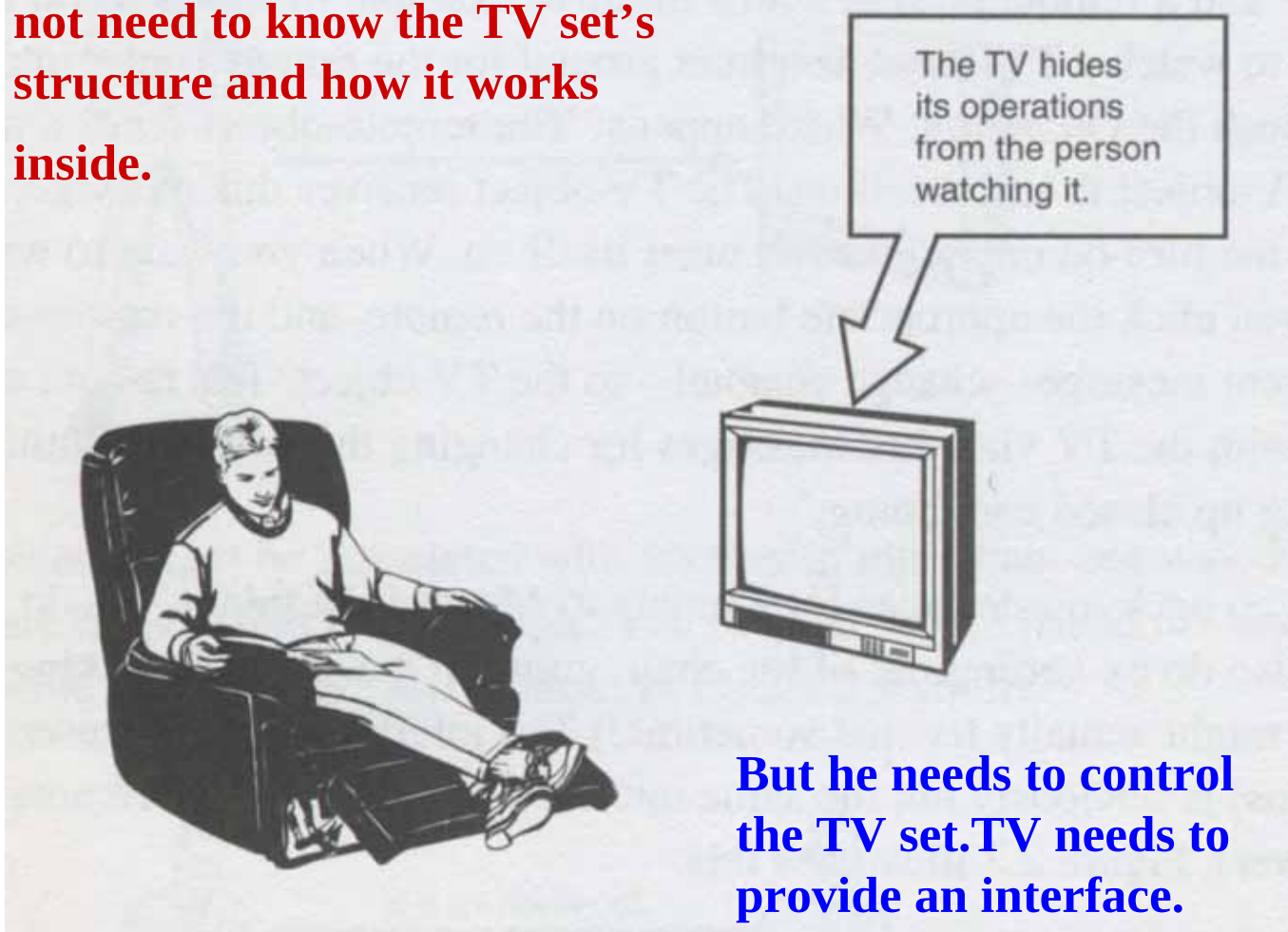
Interface and Implementation



Interface and Implementation

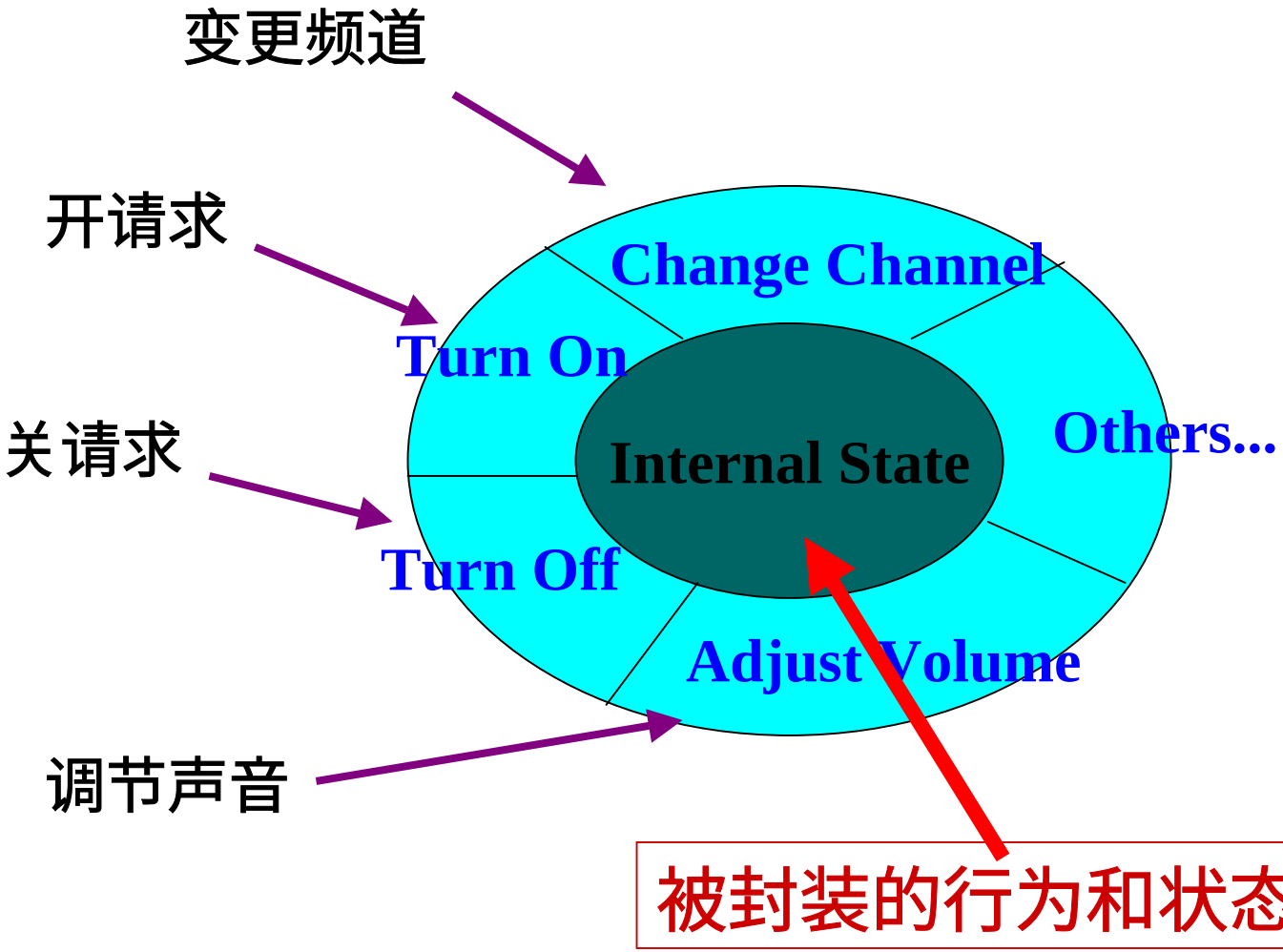



The person watching TV does not need to know the TV set's structure and how it works inside.

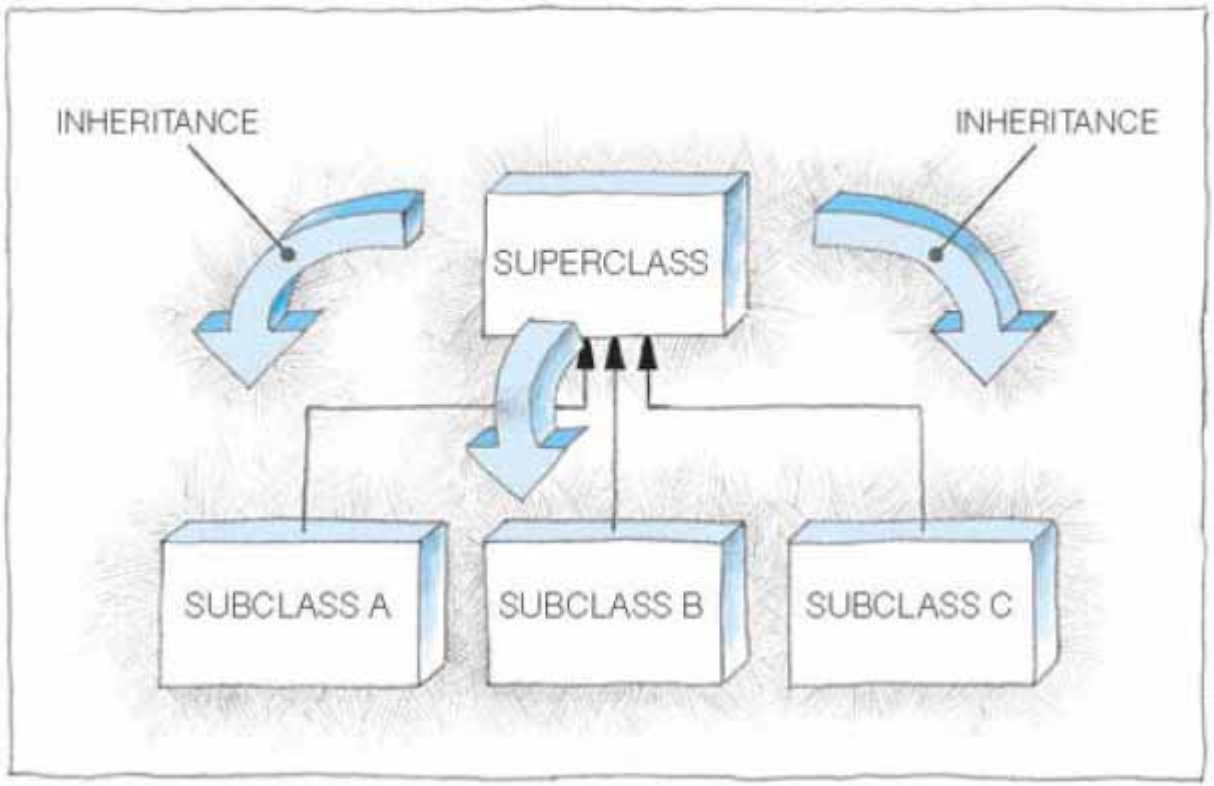


But he needs to control the TV set.TV needs to provide an interface.

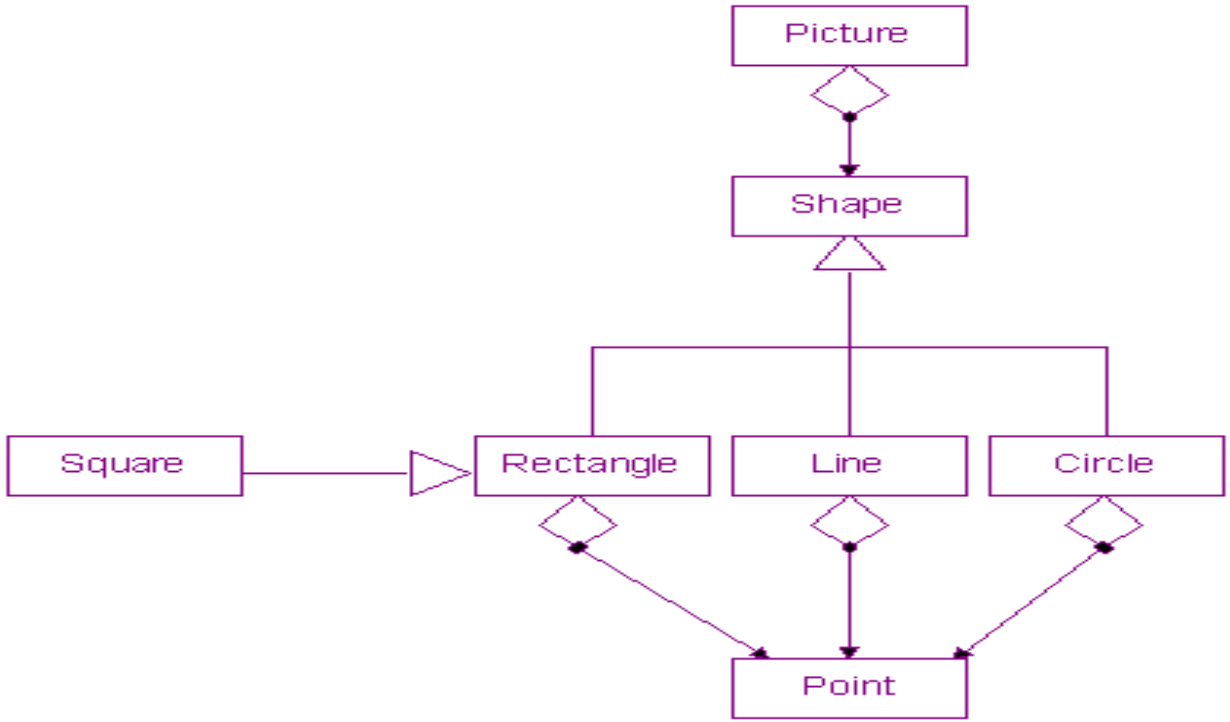
TV


变更频道
开请求
关请求
调节声音
Change Channel
Turn On
Others...
Internal State
Turn Off
Adjust Volume
被封装的行为和状态

Inheritance



Inheritance



OOP with Java

- Access Modifiers
  - **private** Makes a method or a variable accessible only from within its own class.
  - **protected** Makes a method or a variable accessible only to classes in the same package or subclasses of the class.
  - **Public** Makes a class, method, or variable accessible from any other class.
- Class, Method, Variable Modifiers
  - **abstract** Used to declare a class that cannot be instantiated, or a method that must be implemented by a nonabstract subclass.
  - **class** Keyword used to specify a class.
  - **extends** Used to indicate the superclass that a subclass is extending.
  - **implements** Used to indicate the interfaces that a class will implement.
  - **interface** Keyword used to specify an interface.
  - **new** Used to instantiate an object by invoking the constructor.

Declaring Java Classes

- Basic syntax of a Java class

```
<class_declaration> ::=
    <modifier> class <name> {
        <attribute_declaration>*
        <constructor_declaration>*
        <method_declaration>*
}
```

- Example:

```
public class Vehicle {
    private double maxLoad;
    public void setMaxLoad(double value) {
        maxLoad = value;
    }
}
```

Declaring Attributes

- Basic syntax of an attribute

```
<attribute_declaration> ::=
  <modifier> <type> <name> [= <default_value>];

<type> ::= byte | short | int | long | char | float | double |
           boolean | <class>
```

- Examples

```
public class Foo {
    public int x;
    private float y = 10000.0F;
    private String name = “Hello”;
}
```

Declaring Methods

- Basic syntax of a method

```
<method_declaration> ::=
    <modifier> <return_type> <name> (<parameter>*) {
    <statement>*
}
```

- Examples:

```
public class Thing {
    private int x;
    public int getX() {
        return x;
    }
    public void setX(int newX) {
        x = newX;
    }
}
```

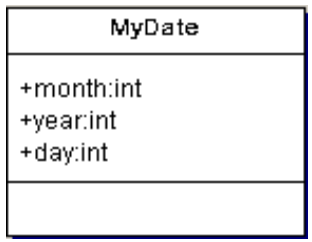
Accessing Object Members

- The “dot” notation <object>, <member>
- This is used to access object members including attributes and methods
- Examples

```
thing1.setX(47);
thing1.x = 47; // only permissible if x is public
```

Information Hiding

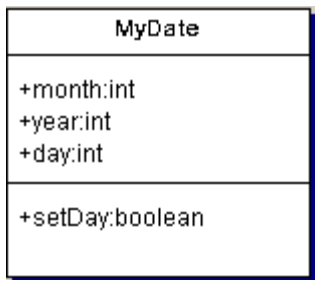
- The problem



Client code has direct access to internal data

```
MyDate d = new MyDate()

d.day = 32
// invalid day
```

- The solution



Client code must use setters/getters to access internal data

```
d.setDate(32)
// invalid day , return false
```

Encapsulation

- Hide the implementation details of a class
- Forces the user to use to interface to access data
- Make the code more maintainable

MyDate
-day : int -month : int
-getDay() : long -setDay() : int -getMonth() : int -setMonth() : int

Declaring Constructor

- Basic syntax of a constructor

```
<constructor_declaration> ::=
      <modifier> <class_name> (<parameter>*) {
      <statement>*
 }
```

- Examples:

```
public class Thing {
      private int x;
      public Thing() {
            x = 47;
      }
       public Thing(int new_x) {
            x = new_x;
       }
}
```

Declaring constructor

- A constructor is a set of instructions designed to initialize an instance. Parameters can be passed to the constructor in the same way as for a method.
- The name of the constructor must always be the same as the class name.
- Constructors are not methods. The do not have return values and are not inherited.

Declaring constructor – For example

```
public class Thing {
    private int x;

    public Thing() {
        x = 47;
    }

    public Thing(int newX) {
        x = newX;
    }

    public int getX() {
        return x;
    }

    public void setX(int newX) {
        x = newX;
    }
}
```

```
public class TestThing {

    public static void main(String[]
     args) {
        Thing thing1 = new  Thing();
        Thing thing2 = new Thing(42);
        System.out.println("thing1.x is “
            + thing1.getX());
        System.out.println("thing2.x is “
            + thing2.getX());
    }
}
```

The output is:

```
thing1.x is 47
thing2.x is 42
```

The Default Constructor

- There is always at least one constructor in every class
- If the writer doest not supply any constructors, the default constructor will be present automatically
  - The default constructor takes no arguments
  - The default constructor has no body
- Enables you to create object instances with new Xxx() without have to write a constructor

The package Statement

- Basic syntax of the package statement

```
<package_declaration> ::=
  package <top_pkg_name>[.<sub_pkg_name>]*;
```

- Example:

```
package org.huihoo.jfox;
```

- Specify the package declaration at the beginning of the source file
- Only one package declaration per source file
- If no package is declared, then the class “belongs” to the default package
- Package names must be hierarchical and separated by dots

The import Statement

- Basic syntax of the package statement

```
<import_declaration> ::=
    import <pkg_name>[.<sub_pkg_name>]*.<class_name |
*>;
```

- Examples:

```
import org.huihoo.jfox.*;
import java.util.List;
import java.io.*;
```

- Precedes all class declarations
- Tells the compiler where to find classes to use

Terminology Recap

- **Class** – A way to define new types in the Java programming language. The class can be considered as a blueprint – a model of the object you are describing.
- **Object** – An actual instance of a class. An object is what you get each time you instantiate a class using new. An object is also known as an instance
- **Attribute** – A data element of an object. An attribute stores information for an object. An attribute is also known as a data member, an instance variable, or a data field
- **Method** – A functional element of an object. A method is also known as a function or a procedure.
- **Constructor** – A “method-like” construct used to initialize (or build) a new object. Constructors are not members (for examples, they are not inherited)
- **Package** – A grouping of classes and/or subpackages

Using Java API Documents

- A set of html files providers information about the API
- One package contains hyperlinks to information on all of the classes
- A class document includes the class hierarchy, a description of the class, a list of member variables, a list of constructors, and so on

Using JavaDoc

Java™ 2 Platform
Std. Ed. v1.3.1

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font

All Classes

AbstractAction
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeC
AbstractDocument.Content
AbstractDocument.ElementEdi
AbstractLayoutCache
AbstractLayoutCache.NodeDim
AbstractList
AbstractListModel

Overview Package Class Use Tree Deprecated Index Help *Java™ 2 Platf*
PREV NEXT FRAMES NO FRAMES *Std. Ed. v1.*

Java™ 2 Platform, Standard Edition, v 1.3.1 API Specification

This document is the API specification for the Java 2 Platform, Standard Edition, version 1.3.1.

See:

Description

Java 2 Platform Packages	
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
	Drag and Drop is a direct

Check your progress

- Define modeling concepts: abstraction, encapsulation and packages
- Define class, member, attribute, method, constructor, and package
- Use the access modifiers private and public as appropriate for the guidelines of encapsulation
- Invoke a method on particular object
- In a Java technology program, identify the following:
  - The package statement
  - The import statement
  - Classes, methods, and attributes
  - Constructors
- Use the Java Technology application programming interface (API) online documentation

Primitive Types

- The Java programming language defines eight primitive types:
  - **Logical** boolean
  - **Textual** char
  - **Integral** byte, short, int, and long
  - **Floating** double and float

Logical-boolean

- The boolean data type has two literals, true and false.

- For example, the statement:

  boolean truth = true;

  declares the variable truth as boolean type and assigns it a value of true.

Textual – char and String

char

- Represents a 16-bit Unicode character
- Must have its literal enclosed in single quotes(‘’)
- Uses the following notations:

  ‘a’ The letter a

  ‘\t’ A tab

String

- Is not a primitive data type; it is a class
- Has its literal enclosed in double quotes(“ “)

  “The quick brown fox jumps over the lazy dog.”
- Can be used as follows:

```
String greeting = “Good Morning !! \n”;
String errorMessage = “Record Not Found !”;
```

Integral – byte, short, int, and long

- Uses three forms – Decimal, octal, or hexadecimal

2	The decimal value is two
077	The leading zero indicates an octal value
0xBAAC	The leading 0x indicates a hexadecimal value

- Has a default int
- Defines long by using the letter L or l

Integral – byte, short, int, and long

- The size and range for the four integral types are show in following table

Integer Length	Name or Type	Range
8 bits	byte	-2*7 to 2*7 -1
16 bits	short	-2*15 to 2*15 -1
32 bits	int	-2*31 to 2*31 -1
64 bits	long	-2*63 to 2*63 -1

Floating Point – float and double

- Default is double
- Floating point data types have the following ranges

Float Length	Name or Type
32bits	float
64bits	double

Java Reference Types

- Beyond primitive types all others are reference types
- A reference variable contains a “handle” to an object
- Example:

```
public class MyDate {
    private int day = 1;
    private int month = 1;
    private int year = 2000;
}

public class TestMyDate {
     public static void main(String[] args) {
          MyDate today = new MyDate();
     }
}
```

Constructing and Initializing Object

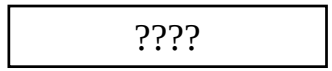
- Calling new Xxx() to allocate space for the new object results in:
  - Memory Allocation: Space for the new object is allocated and instance variables are initialized to their default values
  - Explicit attribute initialization is performed
  - A constructor is executed
  - Variable assignment is made to reference the object
- Example

```
MyDate my_birth = new  MyDate(22, 7, 1964);
```

Memory Allocation and Layout

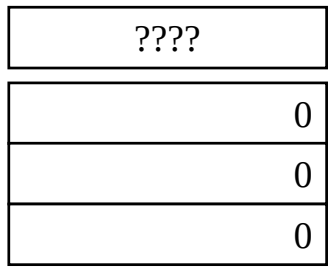
- A declaration allocates storage only for a reference

```
MyDate myBirth = new MyDate(23, 7, 1964);
```



- Use the new operator to allocate space for MyDate:

```
MyDate myBirth = new MyDate(22, 7, 1964);
```



Explicit Attribute Initialization

- Initialize the attribute:

  MyDate myBirth = new MyDate(22, 7, 1964);

myBirth	????
day	1
month	1
year	2000

- The default values are taken from the attribute declaration in the class

Executing the Constructor

- Execute the matching constructor:

```
MyDate myBirth = new MyDate(22, 7, 1964);
```

myBirth	????
day	22
month	7
year	1964

- In the case of an overloaded constructor, the first constructor may call another

Variable Assignment

- Assign newly created object to reference variable

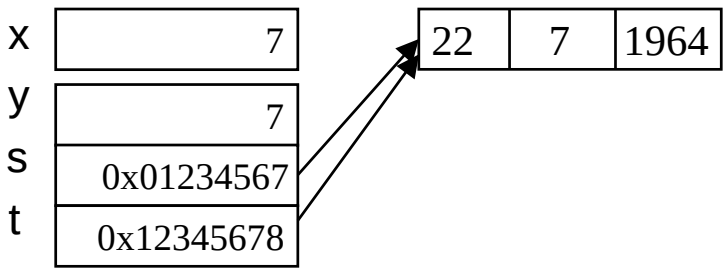
```
MyDate myBirth = new MyDate(22, 7, 1964)
```

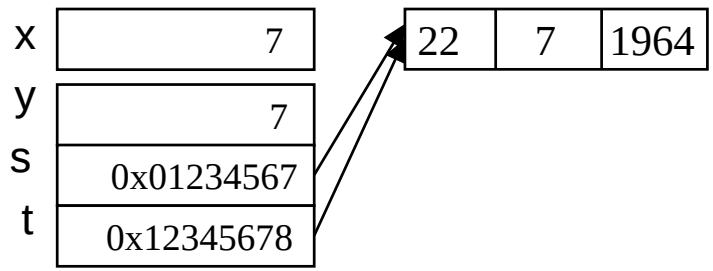
myBirth	????
day	22
month	7
year	1964

Assignment of Reference Variables

- Consider the following code fragment:

```
int x = 7;
int y = x;
MyDate s = new MyDate(22, 7, 1964);
MyDate t = s;
t = new MyDate(22, 12, 164);
```





Pass by Value

- The Java programming language only passes arguments by value
- When an object instance is passed as an argument to a method, the value of the argument is a reference to the object
- The contents of the object can be changed in the called method, but the object reference is never changed

PassTest.java MyDate.java

The this Reference

Here are a few uses of the this keyword

- To reference local attribute and method members within a local method or constructor
  - This is used to disambiguate a local method or constructor variable from an instance variable

- To pass the current object as a parameter to another method or constructor

- Example:

```
public MyDate(int day, int month, int year) {
    this.day = day;
    this.month = month;
    this.year = year;
}
```

Java Coding Conventions

- Packages

  package banking.object
- Classes:

  class SavingsAccount
- Interfaces:

  interface Account
- Methods:

  balanceAccount()
- Variables:

  currentCustomer
- Constants:

  HEAD_COUNT

References

- *Java Coding Style* *Achut Reddy* Server Management Tools Group Sun Microsystems, Inc.

http://java.sun.com/docs/codeconv/

- *Writing Robust Java Code* The AmbySoft Inc. Coding Standards for Java

http://www.ambysoft.com/javaCodingStandards.html

Check Your Progress

- Recognize Java technology keywords
- List the eight primitive types
- Define literal values for numeric and textual types
- Define the terms primitive variable and reference variable
- Declare variables of class type
- Construct an object using new
- Describe default initialization
- Describe the significance of a reference variable
- Java code convention

Inheritance

Upon completion of this module, you should be able to:

- Define inheritance, polymorphism, overloading, overriding
- Describe constructor and method overloading
- In a Java program, identify the following:
  - Overloaded methods and constructors
  - The use of this to call overloaded constructors
  - Overridden methods
  - Invocation of super class methods
  - Parent class constructors
  - Invocation of parent class constructors

Inheritance relationship

- The Employee class

Employee
-name : String -salary : double
-getDetail : String

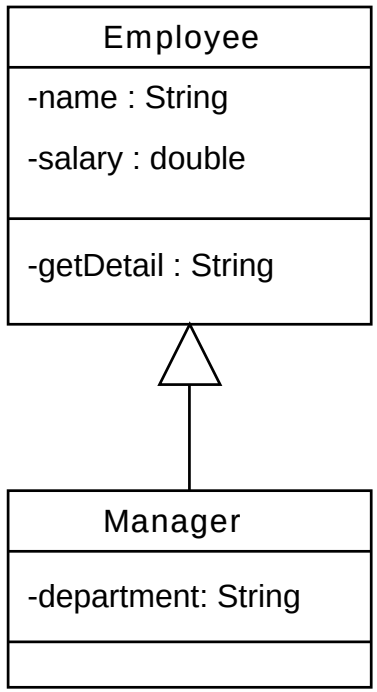
```
public class Employee {
      public String name = “”
      public double salary;


      public String getDetails() { … }
}
```

- The Manager class

Employee
-name : String -salary : double -department : String
-getDetail : String

```
public class Manager {
      public String name = “”
      public double salary;
      public String department = “”

      public String getDetails() { … }
}
```

Inheritance relationship



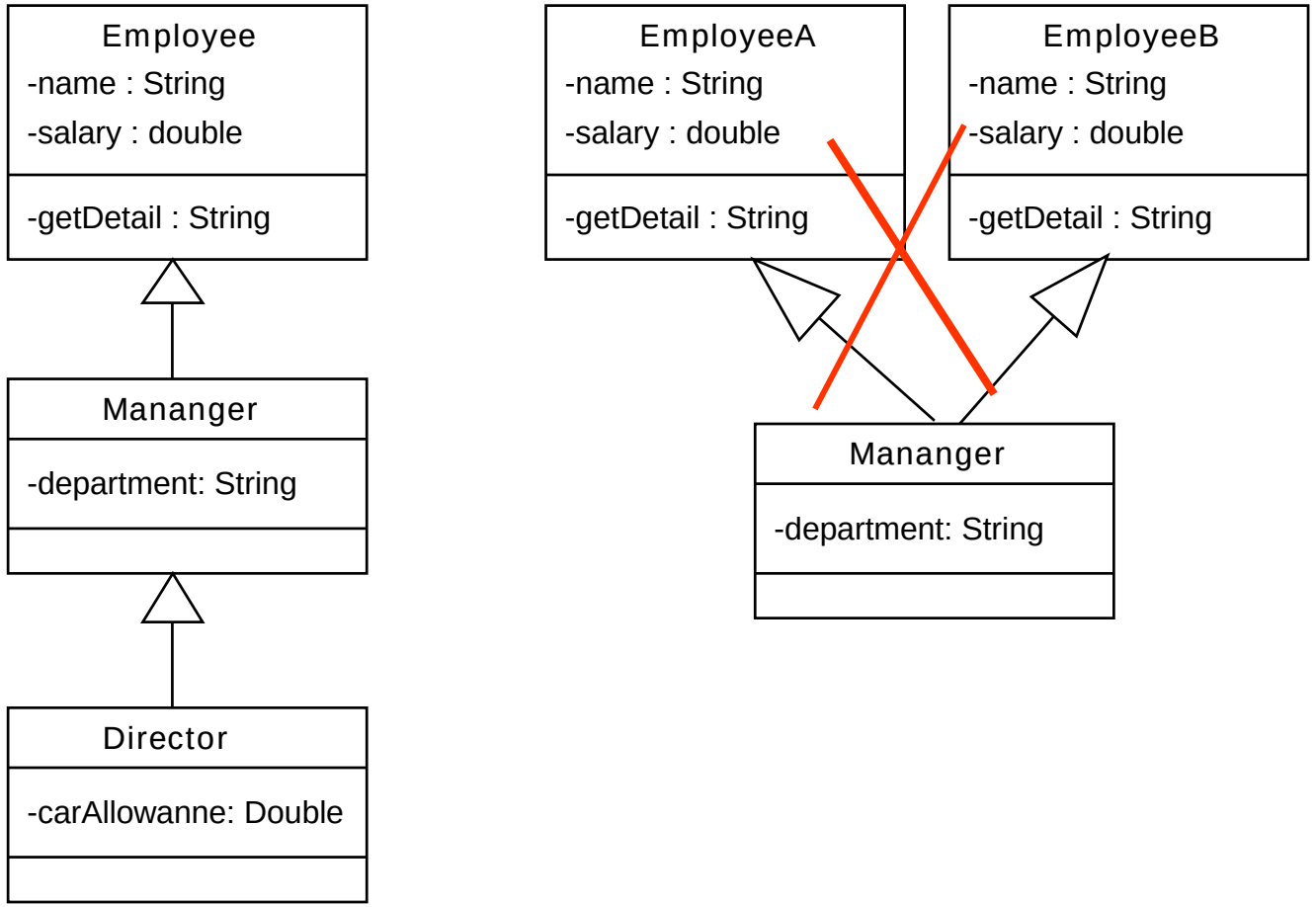

```
public class Employee {
        public String name = “”
        public double salary;

        public String getDetails() { ... }
}

public class Employee extends Employee {
        public String department = “”;
}
```

Single Inheritance

- When a class inherits from only one class, it is called *single inheritance*
- Single inheritance makes code more reliable
- Interfaces provide the benefits of multiple inheritance without drawbacks.
- Syntax of a Java class:

```
<class_declaration>   ::=
      <modifier> class <name>[extends <superclass>] {
         <declarations>*
}
```

Single Inheritance



Constructors Are Not Inherited

- A subclass inherits all methods and variables from the supperclass (parent class)
- A subclass does not inherit the constructor from the superclass
- Two ways to include a constructor are:
  - Use the default constructor
  - Write one or more explicit constructors

Polymorphism

- *Polymorphism* is the ability to have many different forms; for example, the Manager class has access to methods from Employee class
- An object has only one form
- A reference variable can refer to objects of different forms

Polymorphism-The instanceof Operator

- If you receive an object using a reference of type Employee, it might turn out to be a Manager or an Engineer. You can test it by using instanceof as follows

```
public void doSomething(Employee e) {
    if (e instanceof Manager) {
        // Process a Manager
    } else if (e instanceof Engineer) {
        // Process a Engineer
    } else {
        // Process any other type of Employee
    }
}
```

Casting Objects

- Use instanceof to test the type of an object
- Restore full functionality of an object by casting
- Check for proper casting using the following guidelines:
  - Casts up hierarchy are done implicitly
  - Downward casts must be to a subclass and checked by the compiler
  - The object type is checked at runtime when runtime errors can occur

Casting Objects

- In circumstances where you have received a reference to a parent class, and you have determined that the object is actually a particular subclass by using the instanceof operator, you can restore the full functionality of the object by casting the reference.

```
public void doSomething(Employee e) {
        if (e instanceof Manager) {
            Manager m = (Manager)e;
            System.out.println("This is the manager of" +
                m.getDepartment())
        }
         // rest of operation
}
```

Access control

Modifier	Same Class	Same Package	Subclass	Universe
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	
Private	Yes			
Default	Yes	Yes		

Overloading Method Names

- It can be used as follows

```
public void println(int i)
public void println(float f)
public void println(String s)
```

- Argument lists must differ
- Return types can be different

Overloading Constructors

- As with methods, constructors can be overloaded
- Example:

```
public Bank(String name, double rate, int numberOfCustomer)
public Bank(String name, double rate)
public Bank(String name)
```

- Argument lists must differ
- The this reference can be used at the first line of a constructor to call another constructor

Overriding Methods

- A subclass can modify behavior inherited from a parent class
- A subclass can create a method with different functionality than the parent's method but with the same:
  - Name
  - Return type
  - Argument list

The super keyword

- super is used in a class to refer to its superclass
- super is used to refer to the members of superclass, both data attributes and methods

```
public class Employee {
    private String name;
    private double salary;
    private Date birthDate;
    public String getDetails() {
        return "Name: " + name + "\nSalary: " + salary;
    }
}

public class Manager extends Employee {
    private String department;
    public String getDetails() {
        // call parent method
        return super.getDetails() + "\nDepartment: " + department;
    }
}
```

Check Your Progress

- Define inheritance, polymorphism, overloading, overriding
- Use the access modifiers protected and “package-friendly”
- Describe constructor and method overloading
- Describe the complete object construction and initialization operation
- In a Java program, identify the following:
  - Overloaded methods and constructors
  - The use of this to call overloaded constructors
  - Overridden methods
  - Invocation of super class methods
  - Parent class constructors
  - Invocation of parent class constructors

Think Beyond

- Now that you understand inheritance and polymorphism, how can you use this information on a current or future project?

Exercises

- Using the current Java keywords, write program to create a class and an object from the class. Compile and run the program; then verify that the references are assigned

- Write program to create a superclass and subclass, use overload constructor, overload method name, overriding method, super keyword

Further Reading

- Scott W. Ambler **The object primer 2nd Edition The Application Developer's Guide to object-Orientation and the UML** Cambridge University Press, 2001
- Bertrand Meyer **Object-Oriented Software Construction 2nd** ISE Inc.
- Stephen Gilbert, Bill McCarty **Object-Oriented Design In Java** Sams 2001

Resources

- http://www.epubcn.net
- http://www.patternscentral.com/
- http://www.hillside.net/
- http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html

Q&A




Thank You