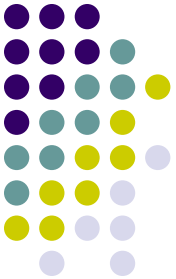
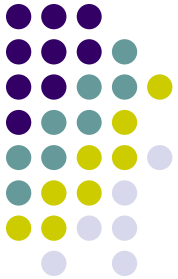
第三章 Java面向对象程序设计 – 2



3.5 类的继承 (Inheritance)

- 类的继承——软件重用的一种方法
  - 一种由已有的类创建新类的机制，是面向对象程序设计的基石之一，也是面向对象技术的三大特性之一。
  - 通过继承，可以根据已有类来定义新类，新类拥有已有父类的所有功能
  - Java只支持类的单继承，每个子类只能有一个直接父类
  - 父类是所有子类的公共属性及方法的集合，子类则是父类的特殊化。
  - 继承机制可以提高程序的抽象程度，提高代码的可重用性

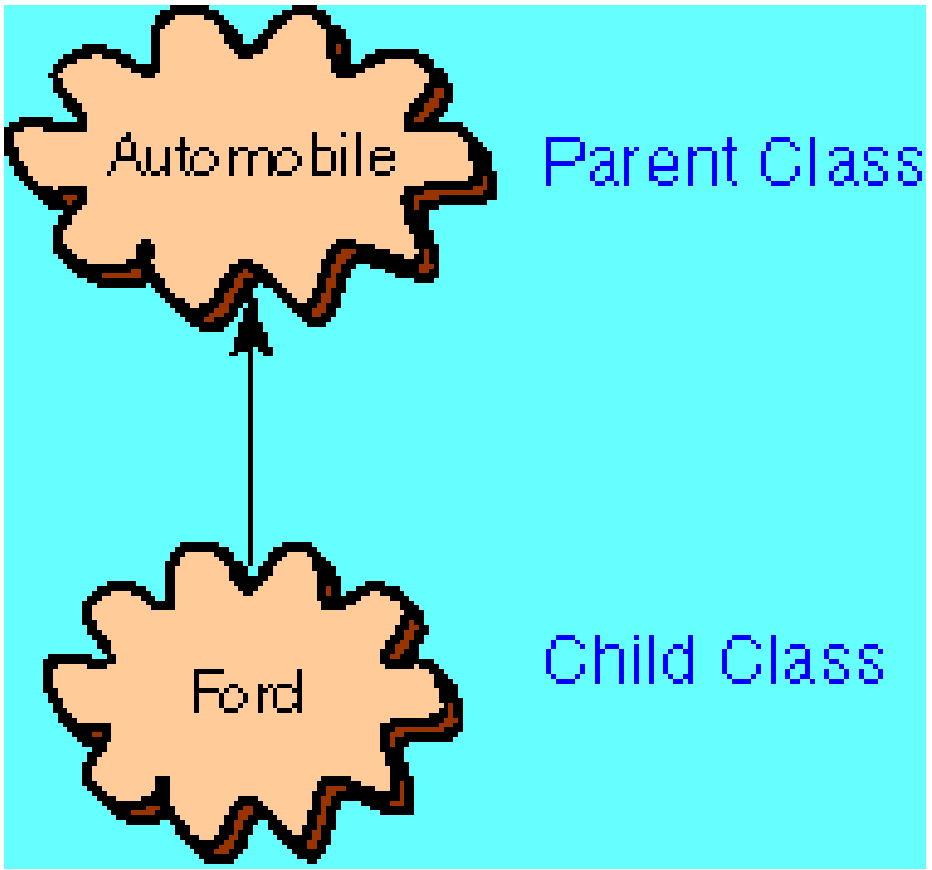
3.5.1 继承的概念



- **基类(base class)**
  - 也称超类(superclass)
  - 是被直接或间接继承的类
- **派生类(derived-class)**
  - 也称子类 (subclass)
  - 继承其他类而得到的类
  - 继承所有祖先的状态和行为
  - 派生类可以增加变量和方法
  - 派生类也可以覆盖(override)继承的方法

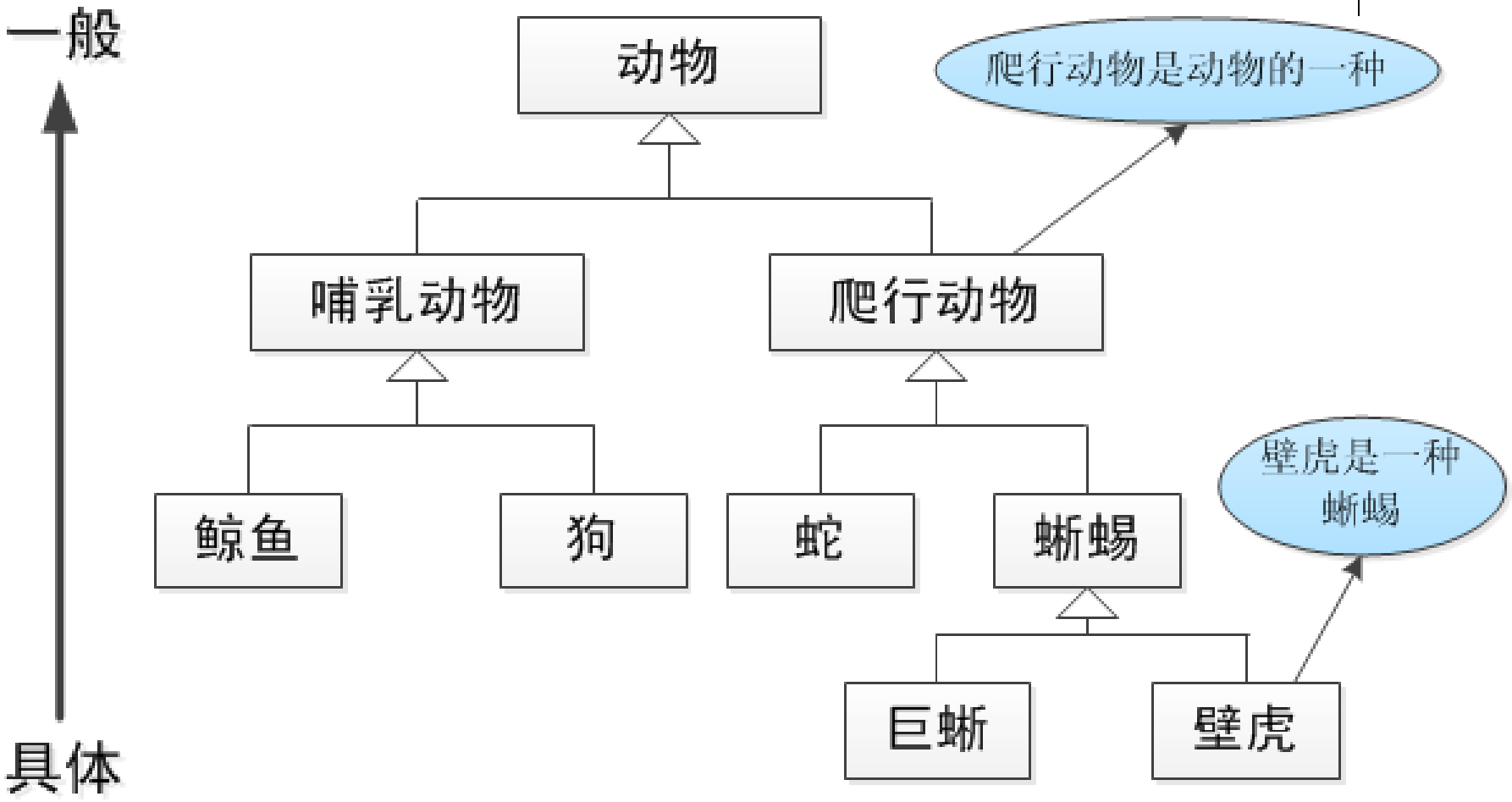
3.5.1 继承的概念

- is_a关系
  - 子类对象与父类对象存在“IS A”(或“is kind of”)的关系



3.5.1 继承的概念

- 动物类层次举例

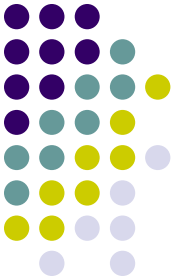


3.5.2 继承的语法

- 继承的语法

```
class childClass extends parentClass
{
    //类体
}
```

3.5.2 继承的语法



举例：

- 在一个公司中，有普通员工（Employees）及管理人员（Magagers）两类人员
- 职员对象（Employees）可能有的属性信息包括
  - 员工号（employeeNumber）
  - 姓名（name）
  - 地址（address）
  - 电话号码（phoneNumber）
- 管理人员（Managers）除具有普通员工的属性外，还可能具有下面的属性
  - 职责（responsibilities）
  - 所管理的职员（listOfEmployees）

3.5.2 继承的语法

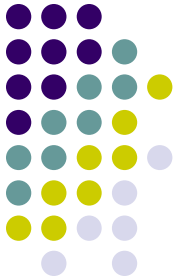
- 父类Employee

```
class Employee
{
    int  employeeNumbe ;
    String  name, address, phoneNumber ;
}
```

- 子类Manager

```
class  Manager extends Employee
{
    //子类增加的数据成员
    String responsibilities, listOfEmployees;
}
```

3.5.2 继承的语法



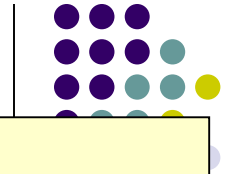
- 设有三个类：Person, Employee, Manager。

```
public class Person {
   public String name;
   public String getName() {
         return name;
   }
}
```

```
public class Employee extends Person {
    public int employeeNumber;
    public int getEmployeeNumber() {
          return employeeNumber;
    }
}
```

```
public class Manager extends Employee {
      public String responsibilities;
      public String getResponsibilities() {
            return responsibilities;
      }
}
```

- 测试上例 ——InheritanceTest.java



```
public class InheritanceTest {
  public static void main(String args[]){
    Employee li = new Employee();
    li.name = "Li Ming";
    li.employeeNumber = 123456;
    System.out.println(li.getName());
    System.out.println(li.getEmployeeNumber());

    Manager he = new Manager();
    he.name = "He Xia";
    he.employeeNumber = 543469;
    he.responsibilities = "Internet project";
    System.out.println(he.getName());
    System.out.println(he.getEmployeeNumber());
    System.out.println(he.getResponsibilities());
  }
}
```

- 运行结果

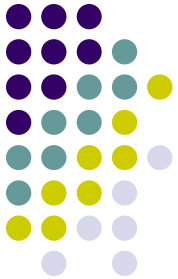
Li Ming
123456
He Xia
543469
Internet project

3.5.2 继承的语法

- 说明
  - 子类不能直接访问从父类中继承的私有属性及方法，但可使用公有（及保护）方法进行访问

```
public class B {
   public int a = 10;
   private int b = 20;
   protected int c = 30;
   public int getB()  { return b; }
}

public class A extends B {
   public int d;
   public void tryVariables() {
      System.out.println(a);          //允许
      System.out.println(b);          //不允许
      System.out.println(getB());  //允许
      System.out.println(c);          //允许
    }
}
```

3.5.3 隐藏和覆盖

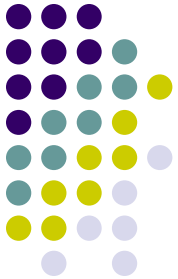


- 隐藏和覆盖
  - 子类对从父类继承来的属性变量及方法可以重新定义
- 属性的隐藏
  - 子类中声明了与父类中相同的成员变量名，则从父类继承的变量将被隐藏
  - 子类拥有了两个相同名字的变量，一个继承自父类，另一个由自己声明
  - 当子类执行继承自父类的操作时，处理的是继承自父类的变量，而当子类执行它自己声明的方法时，所操作的就是它自己声明的变量

```
class Parent {
    int  aNumber;
}
```

```
class Child extends Parent {
    Float aNumber;
}
```

3.5.3 隐藏和覆盖



- 如何访问被隐藏的父类属性
  - 调用从父类继承的方法，则操作的是从父类继承的属性
  - 使用super.属性
- 举例

```
class A1
{
    int x = 2;
    public void setx(int i){
        x = i;
    }
    void printa()
    {
        System.out.println(x);
    }
}
```

```
class B1 extends A1
{
    int x=100;
    void printb()
    {
        super.x = super.x +10 ;
        System.out.println
          ("super.x= " + super.x +
                "  x= " + x);
    }
}
```

3.5.3 隐藏和覆盖

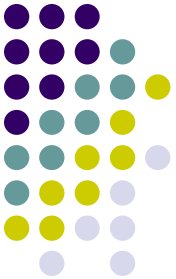
SuperTest1 .java

```
public class SuperTest1
{
    public static void main(String[] args)
    {
      A1 a1 = new A1();
      a1.setx(4);
      a1.printa();

      B1 b1 = new B1();
      b1.printb();
      b1.printa();

      b1.setx(6);  // 将继承到的x值设置为6
      b1.printb();
      b1.printa();
      a1.printa();
    }
}
```

- 运行结果

```
4
super.x= 12  x= 100
12
super.x= 16  x= 100
16
4
```

3.5.3 隐藏和覆盖



- 子类不能继承父类中的静态属性，但可以对父类中的静态属性进行操作。
- 如在上面的例子中，将“int x = 2;”改为“static int x = 2;”，再编译及运行程序。

```
class A1
{
    static int x = 2;
    public void setx(int i){
        x = i;
    }
    void printa()
    {
        System.out.println(x);
    }
}
```

```
class B1 extends A1
{
    int x=100;
    void printb()
    {
        super.x = super.x +10 ;
        System.out.println
          ("super.x= " + super.x +
              "  x= " + x);
    }
}
```

3.5.3 隐藏和覆盖

SuperTest2 .java

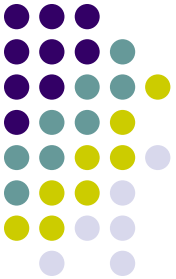
```
public class SuperTest2
{
    public static void main(String[] args)
    {
      A1 a1 = new A1();
      a1.setx(4);
      a1.printa();

      B1 b1 = new B1();
      b1.printb();
      b1.printa();

      b1.setx(6); // 将继承到的x值设置为6
      b1.printb();
      b1.printa();
      a1.printa();
    }
}
```

- 运行结果

  4
  super.x= 14 x= 100
  14
  super.x= 16 x= 100
  16
  16

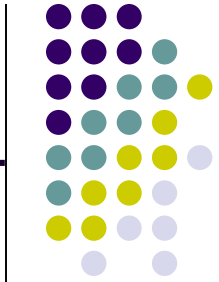
- 在上面的结果中，第一行及最后一行都是语句”a1.printa();”输出的结果，类B中的printb()方法修改的是类 A 中的静态属性x

3.5.3 隐藏和覆盖



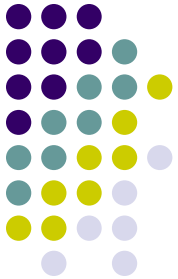
- **方法覆盖——overriding**
  - 如果子类不需使用从父类继承来的方法的功能，则可以声明自己的同名方法，称为**方法覆盖/重写**
  - 覆盖/重写方法的**返回类型，方法名称，参数的个数及类型**必须和被覆盖的方法**一致**
  - 只需在方法名前面**使用不同的类名或不同类的对象名即可区分覆盖方法和被覆盖方法**
  - 覆盖方法的**访问权限**可以比被覆盖的**宽松**，但是不能更为严格

3.5.3 隐藏和覆盖



- 方法覆盖的应用场合
  - 子类中实现与父类相同的功能，但采用不同的算法或公式
  - 在名字相同的方法中，要做比父类更多的事情
  - 在子类中需要取消从父类继承的方法
- 必须覆盖的方法
  - 派生类必须覆盖基类中的抽象的方法，否则派生类自身也成为抽象类.
- **不能覆盖的方法**
  - 基类中声明为final的终结方法
  - 基类中声明为static 的静态方法
- 调用被覆盖的方法
  - super.overriddenMethodName();

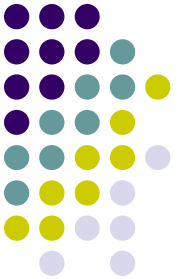
3.5.3 隐藏和覆盖



- super：超类中的变量和方法

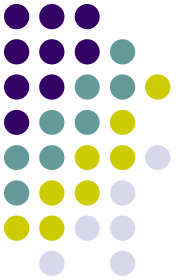
子类成员变量和方法
超类成员变量和方法
} 同名，隐藏超类中

3.5.4 有继承时的构造方法



- 有继承时的构造方法遵循以下原则
  - 子类不能从父类继承构造方法
  - 好的程序设计方法是在子类的构造方法中调用某一个父类构造方法，调用语句必须出现在子类构造方法的第一行，可使用super关键字
  - 如子类构造方法的声明中没有明确调用父类构造方法，则系统在执行子类的构造方法时会自动调用父类的默认构造方法（即无参的构造方法）

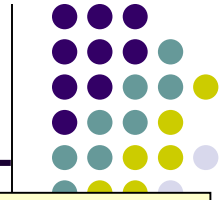
3.5.4 有继承时的构造方法



- 举例

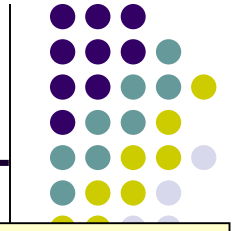
```
public class Person
{
  protected String name, phoneNumber, address;
  public Person()
  {
      this("", "", "");
  }
  public Person(String aName, String aPhoneNumber, String anAddress)
  {
      name=aName;
      phoneNumber=aPhoneNumber;
      address=anAddress;
  }
}
```

3.5.4 有继承时的构造方法



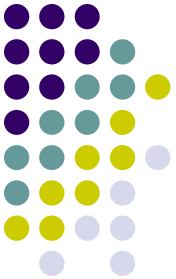
```
public class Employee extends Person
{
    protected int employeeNumber;
    protected String workPhoneNumber;
    public Employee(){
        //此处隐含调用构造方法 Person()
        this(0, "");
    }
    public Employee (int aNumber, String aPhoneNumber){
        //此处隐含调用构造方法 Person()
        employeeNumber=aNumber;
        workPhoneNumber = aPhoneNumber;
    }
}
```

3.5.4 有继承时的构造方法



```
public class Professor extends Employee
{
   protected String research;
   public Professor(){
         super();
         research = "";
   }
   public Professor(int aNumber, String aPhoneNumber, String aResearch)
   {
         super(aNumber, aPhoneNumber);
         research = aResearch;
   }
}
```

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

3.8 abstract类

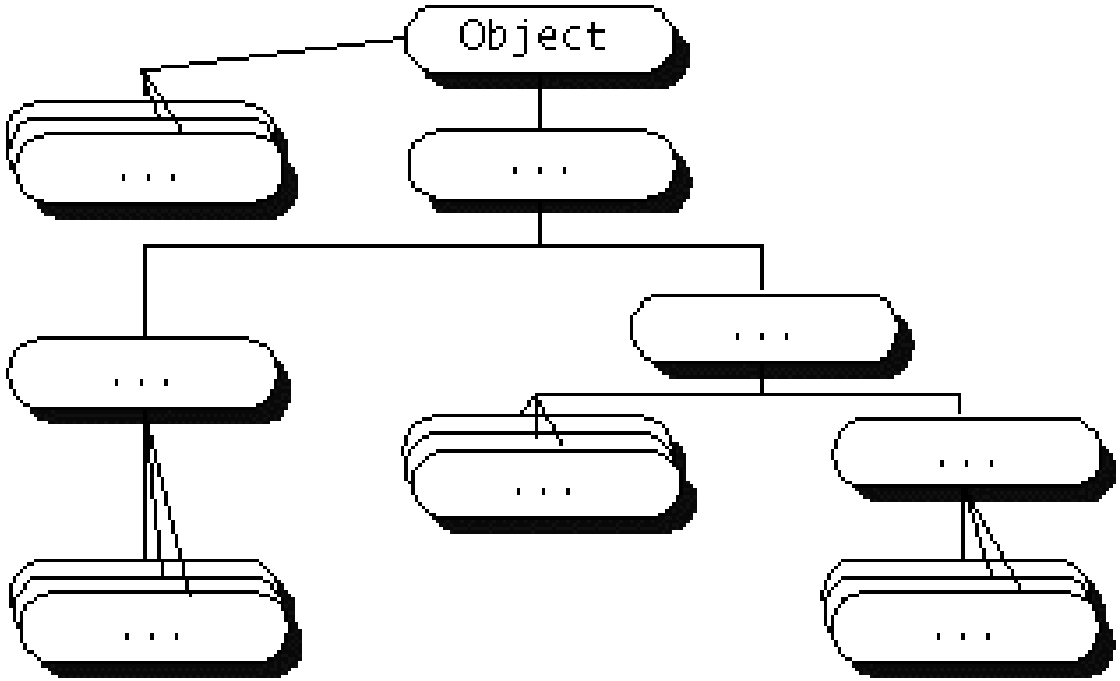
3.9 类的组合

3.10 访问控制

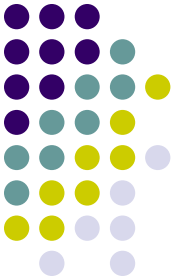
3.11 包

3.6 Object 类

- Object类
  - Java程序中所有类的直接或间接父类，类库中所有类的父类，处在类层次最高点。
  - 包含了所有Java类的公共属性，其构造方法是Object( )

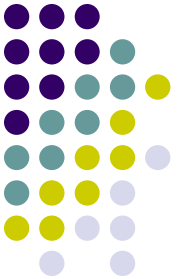


3.6 Object 类



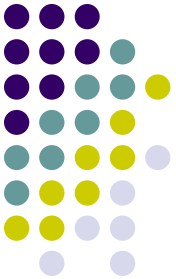
- Object类定义了所有对象必须具有的状态和行为，较主要的方法如下
  - public final Class getClass()
    - 获取当前对象所属的类信息，返回Class对象
  - public String toString()
    - 返回当前对象本身的有关信息，按字符串对象返回
  - public boolean equals(Object obj)
    - 比较两个对象是否是同一对象，是则返回true
  - protected Object clone( )
    - 生成当前对象的一个拷贝，并返回这个复制对象
  - public int hashCode()
    - 返回该对象的哈希代码值
  - protected void finalize() throws Throwable
    - 定义回收当前对象时所需完成的资源释放工作

3.6 Object 类



- 相等和同一的概念
  - 两个对象具有相同的类型，及相同的属性值，则称二者相等(equal)
  - 如果两个引用变量指向的是同一个对象，则称这两个变量(对象)同一 (identical)
  - 两个对象同一，则肯定相等
  - 两个对象相等，不一定同一
  - 比较运算符“==” 判断的是这两个对象是否同一

3.6 Object 类

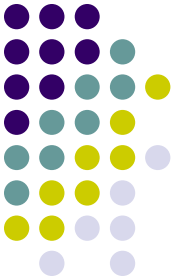


- 判断两个对象是否同一

```
public class EqualTest{
    public static void main(String args[]){
    BankAccount  a = new BankAccount("Bob", 123456, 100.00f);
    BankAccount  b = new BankAccount("Bob", 123456, 100.00f);
    if (a == b)
        System.out.println("YES");
    else
        System.out.println("NO");
  }
}
```

- BankAccount类已声明，此程序运行的结果为“NO”，
- 原因是使用等号“==”判断的是两个对象是否同一，显然a和b是两个对象

3.6 Object 类

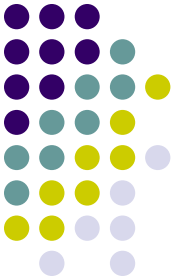


- 修改上程序

```
public class EqualTest{
    public static void main(String args[]){
    BankAccount  a = new BankAccount("Bob", 123456, 100.00f);
    BankAccount  b = a;
    if (a == b)
        System.out.println("YES");
    else
        System.out.println("NO");
  }
}
```

- 将a所指对象的引用赋给b，因此a与b指向的是同一个对象，a与b同一。输出结果为“YES”

3.6 Object 类——equals方法

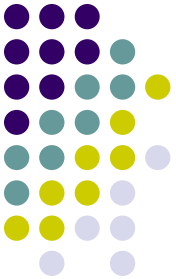


- equals 方法
  - 由于Object是类层次结构中的树根节点，因此所有其他类都继承了equals()方法
  - Object类中的 equals() 方法的定义如下：

```
public boolean equals(Object x) {
    return this == x;
}
```

  - 可见，也是判断两个对象是否同一

3.6 Object 类——equals方法

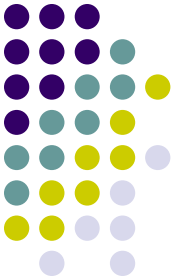


- Object类中equals方法的使用举例

```
public class EqualsTest{
        public static void main(String args[]){
        BankAccount  a = new BankAccount("Bob", 123456, 100.00f);
        BankAccount  b = new BankAccount("Bob", 123456, 100.00f);
        if (a.equals(b))
           System.out.println("YES");
        else
           System.out.println("NO");
    }
}
```

- **由于不是同一对象，运行结果仍然是“NO”**

3.6 Object 类——equals方法



equlas方法重写

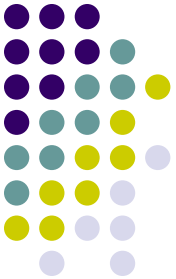
- 要判断两个对象各个属性域的值是否相同，则不能使用从Object类继承来的equals方法，而需要在类声明中对equals方法进行重写
- String类中已经重写了Object类的equals方法，可以判别两个字符串是否内容相同

3.6 Object 类——equals方法

- 在Student类中重写equals方法，方法定义头部必须与Object类中的equals方法完全相同

```
public boolean equals(Object x) {
    if (this.getClass( ) != x.getClass( ))
            return false;
    Student  b = (Student) x;
    return
       ((this.getName( ).equals(b.getName( )))
       &&(this.getStudentNumber( ) == b.getStudentNumber( )));
}
```

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

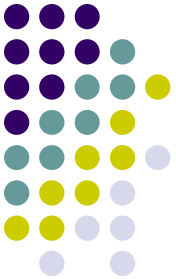
3.8 abstract类

3.9 类的组合

3.10 访问控制

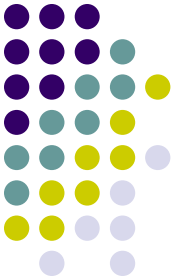
3.11 包

3.7 终结类与终结方法



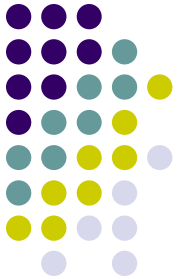
- 终结类与终结方法
  - 被final修饰符修饰的类和方法
  - 终结（final）类不能被继承
  - 终结（final）方法不能被当前类的子类重写

3.7.1 终结类 (final类)



- 终结类的特点
  - 不能有派生类
- 终结类存在的理由
  - 安全: 黑客用来搅乱系统的一个手法是建立一个类的派生类，然后用他们的类代替原来的类
  - 设计: 你认为你的类是最好的或从概念上你的类不应该有任何派生类

3.7.1 终结类 (final类)



- 举例
  - 声明ChessAlgorithm 类为final 类
  - final class ChessAlgorithm { . . . }
- 如果写下如下程序：
  - class BetterChessAlgorithm extends ChessAlgorithm { … }

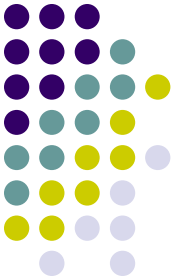
编译器将显示一个错误

```
Chess.java:6: Can't subclass final classes: class ChessAlgorithm
class BetterChessAlgorithm extends ChessAlgorithm {
      ^
1 error
```

3.7.2 终结方法 (final方法)

- 终结方法的特点
  - 不能被派生类覆盖/重写
- 终结方法存在的理由
  - 对于一些比较重要且不希望子类进行更改的方法，可以声明为终结方法。可防止子类对父类关键方法的错误重写，增加了代码的安全性和正确性
  - 提高运行效率。通常，当java运行环境，如java解释器运行方法时，它将首先在当前类中查找该方法，接下来在其超类中查找，并一直沿类层次向上查找，直到找到该方法为止。

3.7.2 终结方法 (final方法)



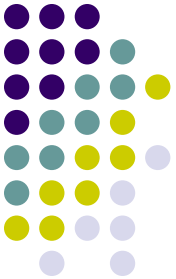
- final 方法举例

```
class Parent
{
   public Parent() {   } //构造方法
   final int getPI() { return Math.PI; } //终结方法
}
```

- 说明
  - getPI()是用final修饰符声明的终结方法，不能在子类中对该方法进行重载。如下声明是错的

```
Class Child extends Parent
{
    public Child() {   }  //构造方法
    int getPI() { return 3.14; } //重写父类中的终结方法，不允许
}
```

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

3.8 abstract类

3.9 类的组合

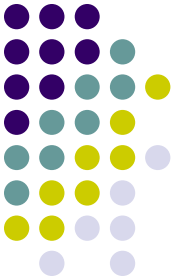
3.10 访问控制

3.11 包

3.8 抽象类(abstract类)

- 抽象类
  - 代表一个抽象概念的类
  - 没有具体实例对象的类，不能使用new方法进行实例化
  - 类前需加修饰符abstract
  - 可包含常规类能够包含的任何东西，例如构造方法，非抽象方法
  - 也可包含抽象方法，这种方法只有方法的声明，而没有方法的实现

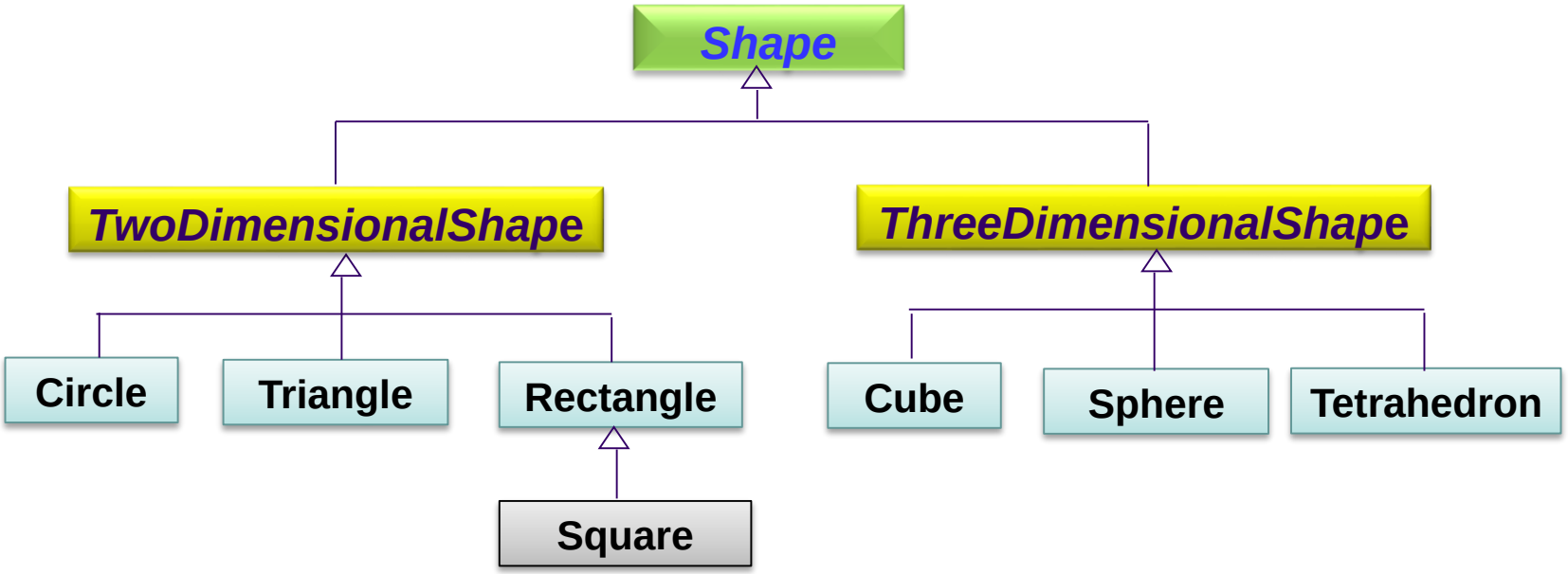
3.8 抽象类(abstract类)



- 存在意义
  - 抽象类是类层次中较高层次的概括，抽象类的作用是让其他类来继承它的抽象化的特征
  - 抽象类中可以包括被它的所有子类共享的公共行为
  - 抽象类可以包括被它的所有子类共享的公共属性
  - 在用户生成实例时强迫用户生成更具体的实例，保证代码的安全性

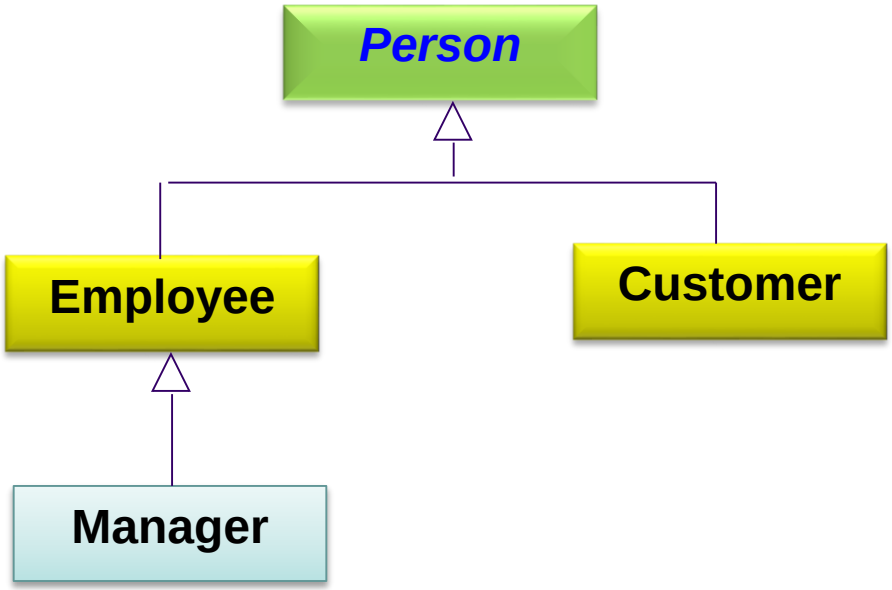
3.8 抽象类(abstract类)

- 将所有图形的公共属性及方法抽象到抽象类Shape。
- 将2D及3D对象的特性分别抽取出来，形成两个抽象类TwoDimensionalShape及ThreeDimensionalShape
  - 2D图形包括Circles、Triangles、Rectangles和Squares
  - 3D图形包括Cube、Sphere、或Tetrahedron



3.8 抽象类(abstract类)

- 举例
  - 如果在应用系统中涉及到的人员只包括：Customers，Employees 及 Managers。则Person类的子类对象覆盖了应用中的对象，可以将Person类声明为抽象类



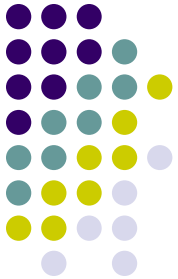
3.8.1 abstract类声明

- 抽象类声明
  - 语法形式为

```
abstract class Number {
    . . .
}
```

如果写： new Number();
编译器将显示错误

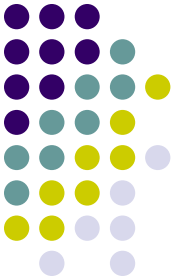
3.8.2 abstract方法



- 抽象方法
  - 声明的语法形式为

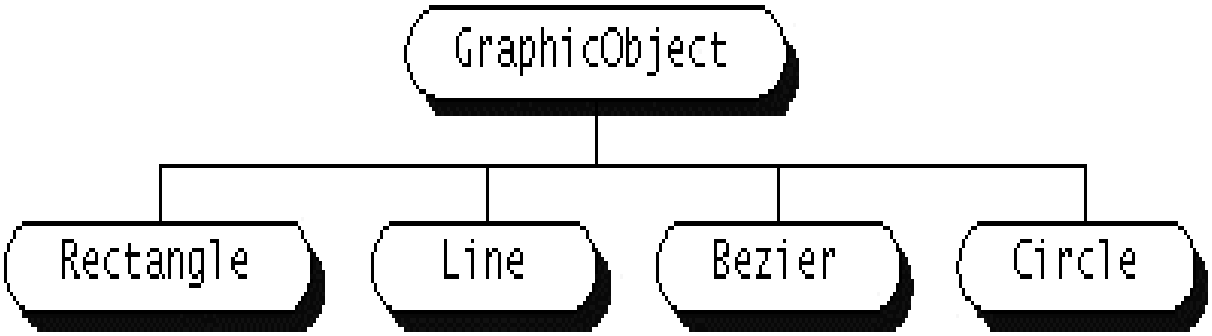
    public abstract <returnType> <methodName>(...);
  - 仅有方法头，而没有方法体和操作实现
  - 具体实现由当前类的不同子类在它们各自的类声明中完成
  - 抽象类可以包含抽象方法
- 需注意的问题
  - 一个抽象类的子类如果不是抽象类，则它必须为父类中的所有抽象方法编写方法体，即重写父类中的所有抽象方法
  - 只有抽象类才能具有抽象方法，即如果一个类中含有抽象方法，则必须将这个类声明为抽象类
  - 除了抽象方法，抽象类中还可以包括非抽象方法

3.8.2 abstract方法



- 抽象方法优点
  - 隐藏具体的细节信息，所有的子类使用的都是相同的方法头，其中包含了调用该方法时需要了解的全部信息
  - 强迫子类完成指定的行为，规定其子类需要用到的“标准”行为

- 绘图实例
  - 各种图形都需要实现绘图方法，可在它们的抽象父类中声明一个draw抽象方法

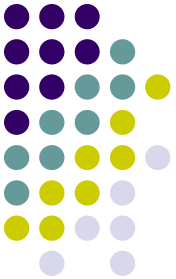



```
abstract class GraphicObject {
    int x, y;
    void moveTo(int newX, int newY) { . . . }
    abstract void draw();
}
```

  - 然后在每一个子类中重

```
class Circle extends GraphicObject {
    void draw() { . . . }
}
class Rectangle extends GraphicObject {
    void draw() { . . . }
}
```

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

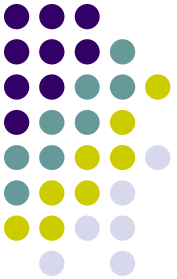
3.8 abstract类

3.9 类的组合

3.10 访问控制

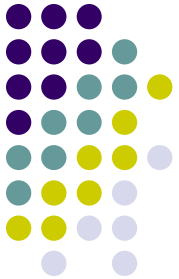
3.11 包

3.9 类的组合(composition)



- 类的组合
  - 现实世界中，大多数对象由更小的对象组成
  - 软件中的对象也常常是由更小的对象组成
  - Java的类中可以有其他类的对象作为成员，这便是类的组合

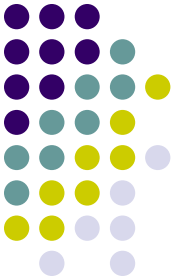
3.9 类的组合



- 组合的语法
  - 组合的语法很简单，只要把已存在类的对象放到新类中即可
  - 可以使用“has a”语句来描述这种关系
  - 例如，考虑Kitchen类提供烹饪和冷藏食品的功能，“my kitchen 'has a' cooker/refrigerator”。
  - 所以，可简单地将对象myCooker和myRefrigerator放在类Kitchen中。格式如下

```
class Cooker{  // 类的语句 }
class Refrigerator{  // 类的语句}
class Kitchen{
    Cooker myCooker;
    Refrigerator myRefrigerator;
}
```

3.9 类的组合



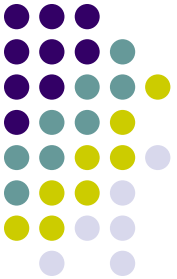
- 举例：一条线段包含两个端点

```
public class Point   //点类
{
    private int x, y;  //coordinate
    public Point(int x, int y) { this.x = x; this.y = y;}
    public int GetX()  {  return x; }
    public int GetY()  {  return y; }
}
```

3.9 类的组合

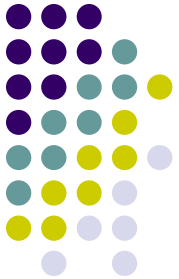
```
import java.math.*;
public class Line   //线段类
{
   private Point  p1,p2;     // 两端点
   Line(Point a, Point b) {
      p1 = new Point(a.GetX(), a.GetY());
      p2 = new Point(b.GetX(), b.GetY());
   }
    public double Length() {
      return Math.sqrt(Math.pow(p2.GetX() - p1.GetX(), 2)
                       + Math.pow(p2.GetY() - p1.GetY(), 2));
    }
}
```

3.9 类的组合



- 组合与继承的比较
  - “包含”关系用组合来表达
    - 如果想利用新类内部一个现有类的特性，而不想使用它的接口，通常应选择组合，我们需在新类里嵌入现有类的private对象
    - 如果想让类用户直接访问新类的组合成分，需要将成员对象的属性变为public
  - “属于”关系用继承来表达
    - 取得一个现成的类，并制作它的一个特殊版本。通常，这意味着我们准备使用一个常规用途的类，并根据特定需求对其进行定制

3.9 类的组合——组合与继承的比较



- Car（汽车）对象，由于汽车的装配是故障分析时需要考虑的一项因素，所以有助于客户程序员理解如何使用类，而且类创建者的编程复杂程度也会大幅度降低。

```
class Engine {  //发动机类
    public void start() { …… }
    public void rev() { …… }
    public void stop() { …… }
}
```

```
class Wheel { //车轮类
    public void inflate(int psi) {
        …….
    }
}
```

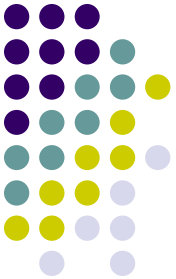
```
class Door { //车门类
    public Window window = new Window();
    public void open() { …… }
    public void close() { …… }
}
```

```
class Window { //车窗类
    public void rollup() {……}
    public void rolldown() {……}
}
```

3.9 类的组合——组合与继承的比较

```
public class Car {
    public Engine engine = new Engine();
    public Wheel[] wheel = new Wheel[4];
    public Window[] wheel = new Window[4];
    public Door left = new Door(),right = new Door();
    public Car() {
            for(int i = 0; i < 4; i++)
                wheel[i] = new Wheel();
     }
    public static void main(String[] args) {
            Car car = new Car();
            car.left.window.rollup();
            Car.wheel[0].inflate(72);
     }
}
```

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

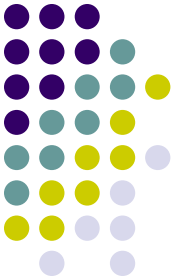
3.8 abstract类

3.9 类的组合

3.10 访问控制

3.11 包

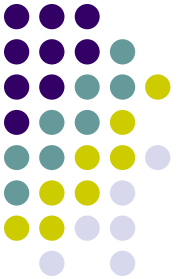
3.10 访问控制



- 类的访问控制
  - 类的访问控制只有两种
    - public（公共类）
    - 无修饰符（缺省类）
  - 访问权限符与访问能力之间的关系如下表

类型	无修饰	public
同一包中的类	yes	yes
不同包中的类	no	yes

3.10 访问控制



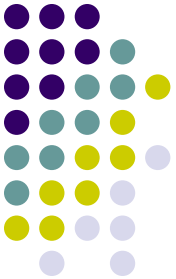
- 类成员的访问控制
  - 公有(public)
    - 可以被其他任何对象访问(前提：有类成员所在的类有访问权限)
  - 保护(protected)
    - 只可被同一类及其子类的实例对象访问
  - 私有(private)
    - 只能被这个类本身访问，在类外不可见
  - 默认(default)
    - 仅允许同一个包内的访问；又被称为“包(package)访问权限”

3.10 访问控制

- 访问权限符与访问能力之间的关系如下表

类型	private	无修饰	protected	public
同一类	yes	yes	yes	yes
同一包中的子类	no	yes	yes	yes
同一包中的非子类	no	yes	yes	yes
不同包中的子类	no	no	yes	yes
不同包中的非子类	no	no	no	yes

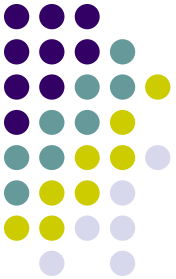
3.10 访问控制



- 举例
  - 对Circle类声明进行修改，给实例变量加上private修饰符

```
public class Circle {
        static double PI = 3.14159265;
        private int radius;
        public double circumference() {
            return 2 * PI * radius;
        }
 }
```

3.10 访问控制

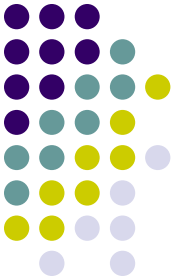


- 测试程序 CircumferenceTester.java

```
public class CircumferenceTester {
    public static void main(String args[]) {
        Circle c1 = new Circle();
        c1.radius = 50;
        Circle c2 = new Circle();
        c2.radius = 10;
        double circum1 = c1.circumference();
        double circum2 = c2.circumference();
        System.out.println("Circle 1 has circumference " + circum1);
        System.out.println("Circle 2 has circumference " + circum2);
    }
}
```

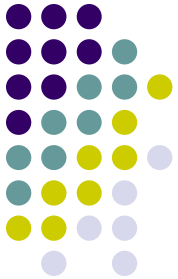
在编译语句“c1.radius = 50;”及“c2.radius =10;”时会提示存在语法错误“radius has private access in Circle”

3.10 访问控制



- 原因
  - 由于在Circle类声明中变量radius被声明为private，因此在其它类中不能直接对radius进行存取
  - 如果要允许其它类访问radius的值，就需要在Circle类中声明相应的公有方法。通常有两类典型的方法用于访问属性值，get方法及set方法。

3.10 访问控制



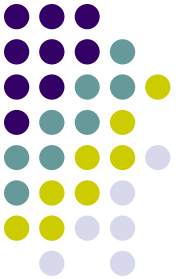
- **get方法**
  - 功能：取得属性变量的值
  - get方法名以“get”开头，后面是实例变量的名字
  - 一般具有以下格式：

```
public <fieldType>  get<FieldName>() {
    return <fieldName>;
}
```

  - 对于实例变量radius，声明其get方法如下：

```
public int getRadius(){
   return radius;
}
```

3.10 访问控制



- set方法
  - 功能：修改属性变量的值
  - set方法名以“set”开头，后面是实例变量的名字
  - 一般具有以下格式

```
public void set<FieldName>(<fieldType> <paramName>) {
    <fieldName> = <paramName>;
}
```

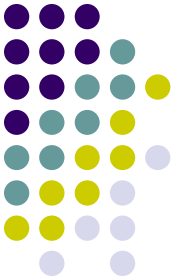
  - 声明实例变量radius的set方法如下：

```
public void setRadius(int r){   radius = r;  }
```

  - 在上面的set方法中，如果形式参数为radius，则需要在成员变量radius之前加上关键字this。代码如下：

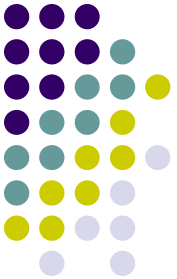
```
public void setRadius(int radius){
   this.radius = radius;
 }
```

3.10 访问控制



- 实际使用
  - public
    - 少量(或者没有) public 域（数据成员）
    - 部分 public 方法
  - private
    - 目的：隐藏具体的实现细节
    - 域常常是private
      - 采用 public “get” 方法➔读取数据
      - 采用 public “set”方法➔写数据

第三章 Java面向对象程序设计 – 2



3.5 类的继承

3.6 Object类

3.7 final类与final方法

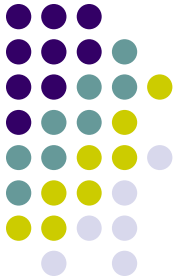
3.8 abstract类

3.9 类的组合

3.10 访问控制

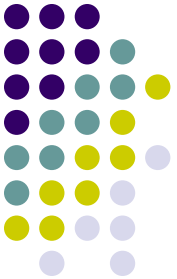
3.11 包

3.11 包 (package)



- 包——类的组织
  - **是一组松散的类集合**
    - 一个包可以包含若干个类文件，还可包含若干个包
  - **包的作用**
    - 将相关的源代码文件组织在一起
    - 类名的空间管理，解决类名冲突
    - Java利用包来组织相关的类，并控制访问权限
    - 利用包来管理类，可实现类的共享与代码复用

3.11.1 包的使用

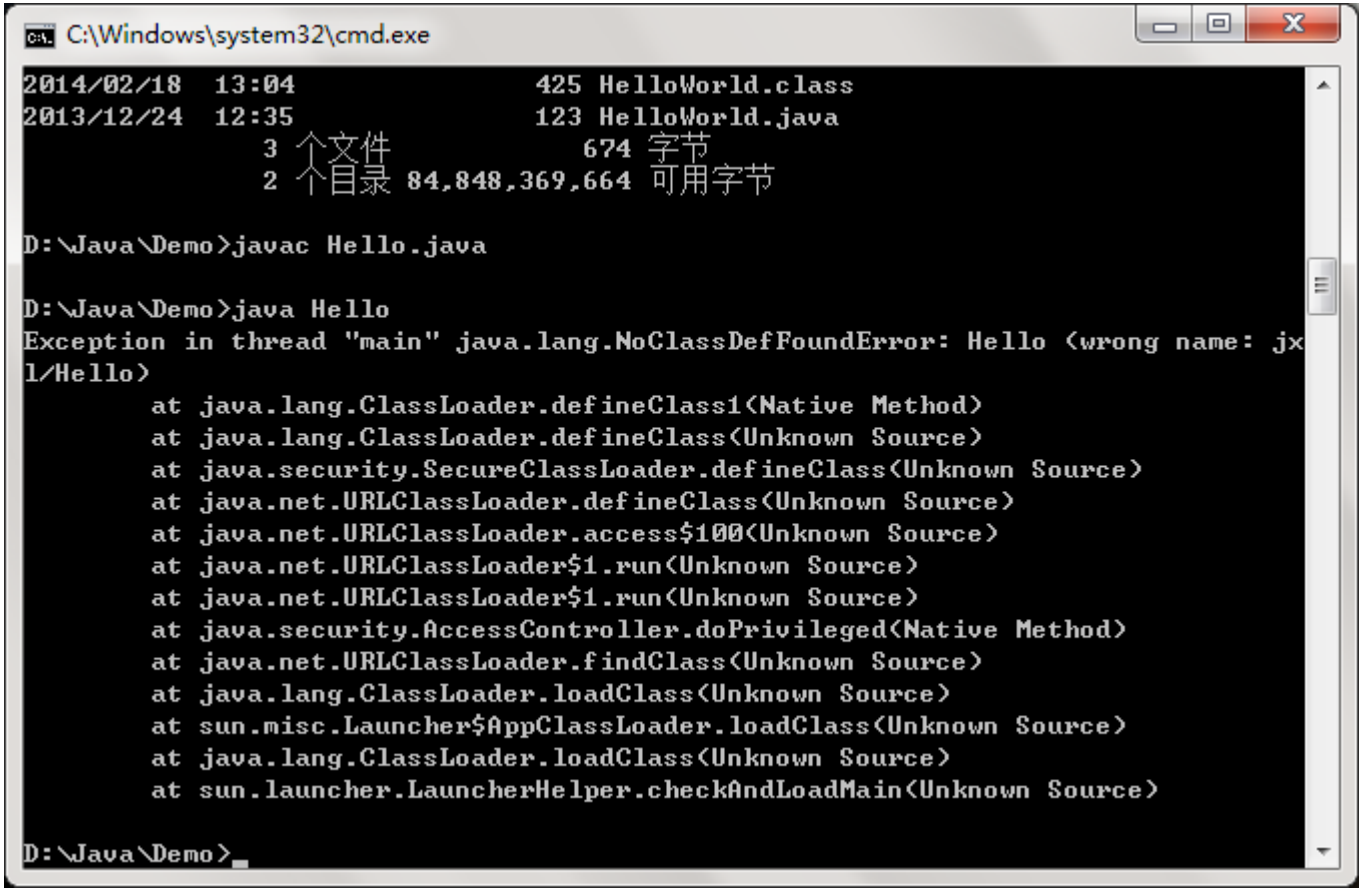


- 使用包
  - 为了使用其它包中所提供的类，需要使用import语句引入所需要的类，如import javax.swing.*;
  - import语句的格式

    import package1[.package2…]. (classname |*);
    - 其中package1[.package2…]表明包的层次，它对应于文件目录
    - classname则指明所要引入的类名
    - 如果要引入一个包中的所有类，则可以使用星号（*）来代替类名
  - Java编译器会将import语句引入的包字符串拼接到标识符前。
  - Java编译器为所有程序自动引入包java.lang

3.11.1 包的使用

- 在源文件中添加package语句可将类加入到指定包中。
- 请使用命令行运行示例：Hello.java



为何生成的.class文件无法运行？

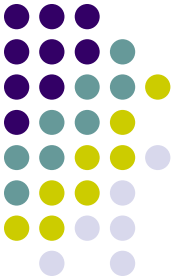
3.11.1 包的使用

- 修正Bug
  - 建立一个包文件夹Julie，将class文件移入此文件夹下

```
D:\Java\Demo>java Hello
Exception in thread "main" java.lang.NoClassDefFoundError: Hello (wrong name: Ju
lie/Hello)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(Unknown Source)
        at java.security.SecureClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.access$100(Unknown Source)
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.launcher.LauncherHelper.checkAndLoadMain(Unknown Source)

D:\Java\Demo>java Julie.Hello
Hello World

D:\Java\Demo>
```

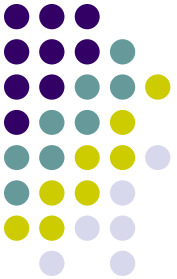
运行一切正常！

3.11.1 包的使用



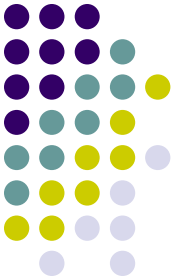
- 编译单元与类空间
  - 一个Java源代码文件称为一个编译单元，由三部分组成
    - 所属包的声明（省略，则属于默认包）
    - import （引入）包的声明，用于导入外部的类
    - 类和接口的声明
  - 一个编译单元中只能有一个public类，该类名与文件名相同，编译单元中的其他类往往是public类的辅助类，经过编译，每个类都会产一个class文件
  - 利用包来划分名字空间，便可以避免类名冲突

3.11.2 自定义包



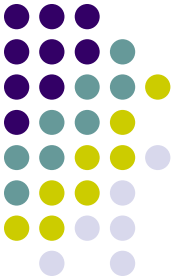
- 自定义包
  - 同一包中的类在默认情况下可以互相访问，通常把需要在一起工作的类放在一个包里
  - 在实际使用中，用户可以将自己的类组织成包

3.11.2 自定义包



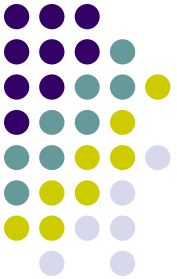
- 包的命名
  - 每个包的名称必须是“独一无二”的
  - Java中包名使用小写字母表示
  - 命名方式建议
    - 将机构的Internet域名反序，作为包名的前导
    - 例如：import java.util.*;
    - cn.edu.hhu.computer.class11
- 包的声明
  - 命名包（Named Packages）
    - package mypackage;
    - 说明当前文件中声明的所有类都属于包mypackage
    - 此文件中的每一个类名前都有前缀mypackage，即实际类名应该是mypackage.ClassName，因此不同包中的相同类名不会冲突
  - 默认包（未命名的包）
    - 不含有包声明的编译单元是默认包的一部分

3.11.2 自定义包



- package语句
  - Java源文件的第一条语句， 前面只能有注释或空行
  - 一个文件中最多只能有一条
  - 如果源文件中没有，则文件中声明的所有类属于一个默认的无名包
  - 包声明的语句的完整格式如下：
    - package pkg1[.pkg2[.pkg3…]];
      - Java编译器把包对应于文件系统的目录结构
      - 用点来指明目录的层次

3.11.2 自定义包



- 包的使用
  - 假设已定义并生成了下面的包

```
package mypackage;
public class MyClass {
  // . . .
}
```

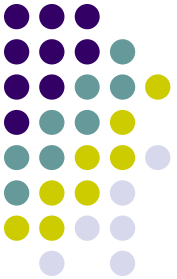
  - 如果其他人想使用MyClass类
    - 使用import语句引入

```
import mypackage.*;
// . . .
MyClass m = new MyClass();
```

    - 不使用import语句，则需要使用全名

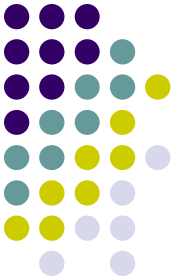
```
mypackage.MyClass m = new mypackage.MyClass();
```

3.11.2 自定义包



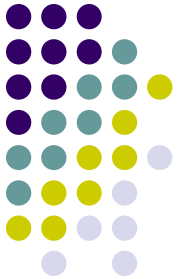
- **编译和生成包**
  - 如果在程序Test.java中已声明了包mypackage
    编译时采用方式 javac -d destpath Test.java
    则编译器会自动在destpath目录下建立子目录mypackage，并将生成的.class文件都放到destpath/mypackage下。
- 举例：手工编写并使用自定义的包
- 演示步骤：
  - 反转一个字串（原型：Test1.java）
  - 将其移为一个函数(Test2.java)
  - 创建类并将其移入一个包中（MyPackageClass.java）
  - 在其它项目中使用此包（Test3.java）

3.11.3 JAR文件



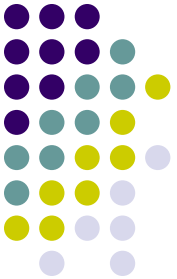
- **路径问题**
  - Java公有类必须放入同名的源文件，编译生成.class文件，如果使用package语句指定了类所属的包名，则.class文件应该放入包所对应的文件夹下。
  - 如果代码中使用了另外包中的类，必须保证此类的.class文件可以在与包名匹配的路径中找到
  - 可以使用CLASSPATH环境变量指明.class文件的查找路径。
  - 为降低复杂程度，可以使用JAR将所有相关文件都打包到同一个压缩包中。

3.11.3 JAR文件



- **什么是JAR**
  - JAR文件是Java 的一种文档格式，其实是一个压缩包，兼容于常见的Zip压缩文件，可以使用WinRAR等软件打开。
  - Jar文件中包容一个清单（manifest）文件，包容一些重要信息。
    - 比如可以在windows资源管理器中“双击”执行的jar包，其清单文件必须指明包中哪个类是“主类（main class）”，从而让JVM知道应该从哪个类中的main方法开始执行。

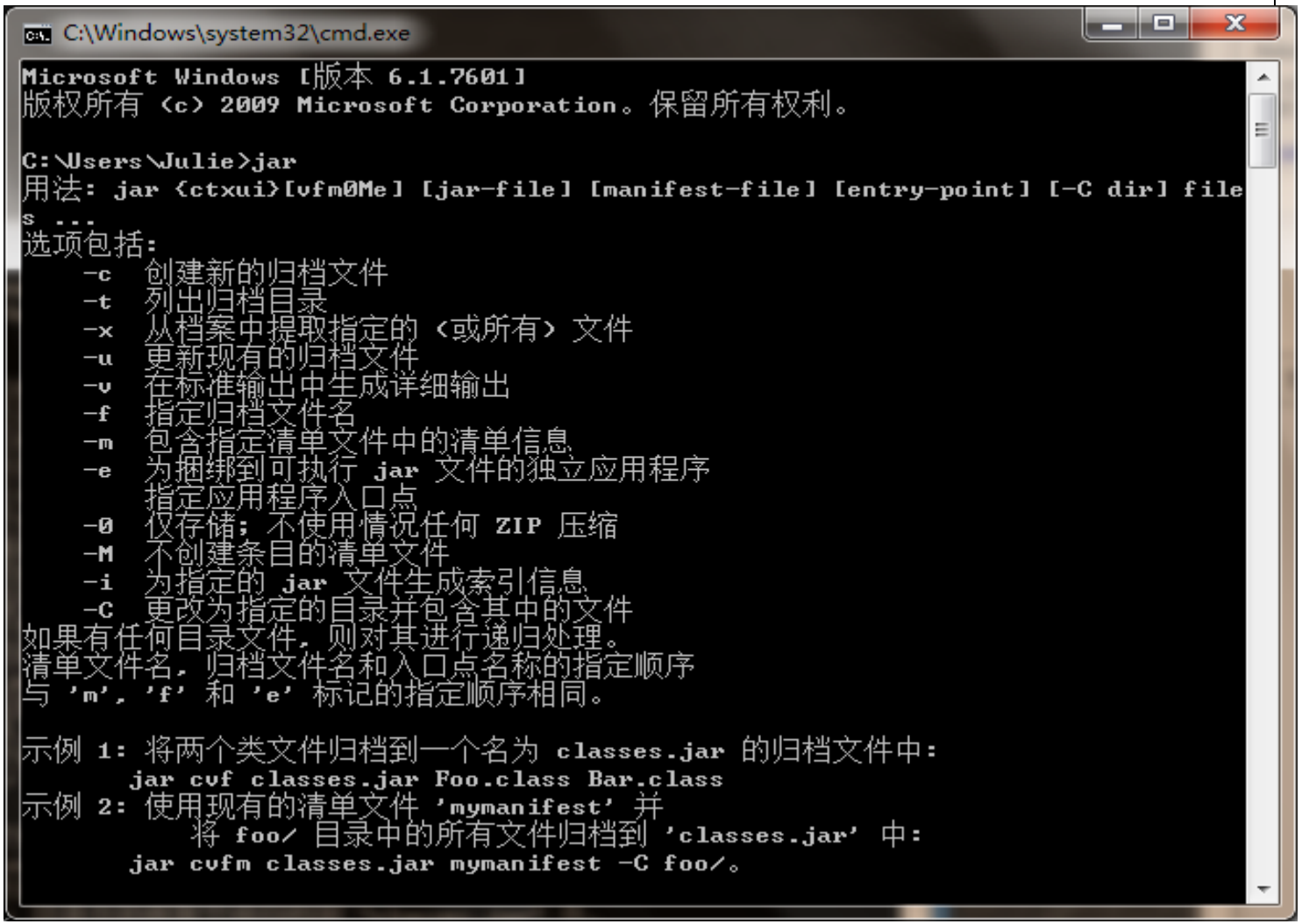
3.11.3 JAR文件



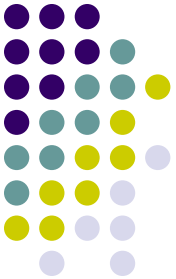
- JAR工具
  - 为了用 JAR 文件执行基本的任务，要使用JDK提供的 jar 工具(Java Archive Tool)
  - 随 JDK 安装，在 JDK 安装目录下的 bin 目录中
    - Windows 下文件名为 jar.exe
    - Linux 下文件名为 jar
  - 它的运行需要用到 JDK 安装目录下 lib 目录中的 tools.jar 文件
  - 用 jar 命令调用

3.11.3 JAR文件

- 了解JAR命令

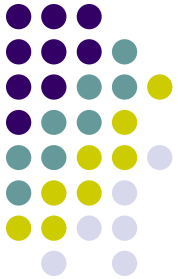


3.11.3 JAR文件



- JAR命令示例
  - jar cf test.jar test
    - 将当前文件夹下test子文件夹的所有内容打包为test.jar
  - jar cvf test.jar test
    - 同前，只不过显示压缩过程
  - jar tf test.jar
    - 查看jar文件内容
  - jar xvf test.jar
    - 解压缩jar包

3.11.4 Java 基础类库简介



- Java基础类库
  - Java提供了用于语言开发的类库，称为Java基础类库(JFC，Java Foundational Class)，也称应用程序编程接口(API，Application Programming Interface)，分别放在不同的包中
  - Java提供的包主要有

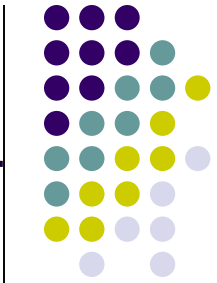
    java.lang，java.io，java.math，java.util

    java.applet，java.awt，java.awt.datatransfer

    java.awt.event，java.awt.image，java.beans

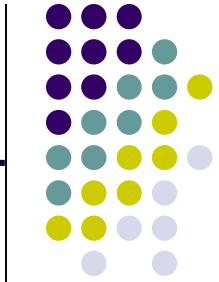
    java.net，java.rmi，java.security，java.sql等

3.11.4 Java 基础类库简介



- 语言包(java.lang)
  - 语言包提供了Java语言最基础的类，包括
    - Object类
    - 数据类型包裹类(the Data Type Wrapper)
    - 字符串类(String、StringBuffer)
    - 数学类(Math)
    - 系统和运行时类(System、Runtime)
    - 类操作类(Class，ClassLoader)

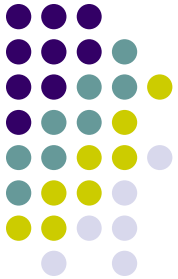
3.11.4 Java 基础类库简介



- 数据包裹类
  - 对应Java的每一个基本数据类型(primitive data type)都有一个数据包裹类
  - 每个包裹类都只有一个类型为对应的基本数据类型的属性域

基本数据类型	数据包裹类
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

3.11.4 Java 基础类库简介



- 生成数据类型包裹类对象的方法
  - 从基本数据类型的变量或常量生成包裹类对象

```
double x = 1.2;
Double a = new Double(x);
Double b = new Double(-5.25);
```

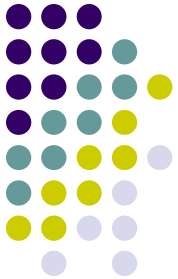
  - 从字符串生成包裹类对象

```
Double c = new Double("-2.34");
Integer i = new Integer("1234");
```

  - 已知字符串，可使用valueOf方法将其转换成包裹类对象：

```
Integer.valueOf("125");
Double.valueOf("5.15");
```

3.11.4 Java 基础类库简介



- 得到基本数据类型数据的方法
  - 每一个包裹类都提供相应的方法将包裹类对象转换回基本数据类型

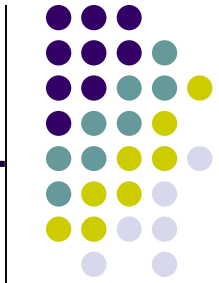
    anIntegerObject.intValue() // 返回 int类型

    aCharacterObject.charValue() // 返回 char类型
  - Integer、Float、Double、Long、Byte 及Short 类提供了方法能够将字符串类型的对象直接转换成对应的int、float、double、long、byte或short类型的数据

    Integer.parseInt(“234”) // 返回int类型的数据

    Float.parseFloat("234.78") // 返回float类型的数据

3.11.4 Java 基础类库简介



- String类
  - 该类字符串对象的值和长度都不变化，为常量字符串
- 生成String类对象的方法
  - 可以这样生成一个常量字符串

    String aString;

    aString = “This is a string”
  - 调用构造方法生成字符串对象

    new String();

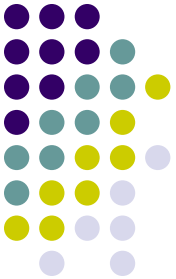
    new String(String value);

    new String(char[] value);

    new String(char[] value, int offset, int count);

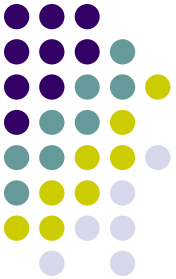
    new String(StringBuffer buffer);
- 后续章节详细介绍

3.11.4 Java 基础类库简介



- 数学类
  - 提供一组常量和数学函数，例如
    - E和PI常数
    - 求绝对值的abs方法
    - 计算三角函数的sin方法和cos方法
    - 求最小值、最大值的min方法和max方法
    - 求随机数的random方法等
  - 其中所有的变量和方法都是静态的(static)
  - 是终结类(final)，不能从中派生其他的新类

3.11.4 Java 基础类库简介



- 系统和运行时类System、Runtime
- System类
  - 访问系统资源
    - arraycopy()　　复制一个数组
    - exit()　　结束当前运行的程序
    - currentTimeMillis()　获得系统当前日期和时间等
  - 访问标准输入输出流
    - System.in　　标准输入，表示键盘
    - System.out　　标准输出，表示显示器
- Runtime类
  - 可直接访问运行时资源
    - totalMemory()　返回系统内存总量
    - freeMemory()　返回内存的剩余空间

内容小结

- 内容
  - 介绍了Java语言类的重用机制，形式可以是组合或继承
  - Object类的主要方法
  - 终结类和终结方法的特点和语法
  - 抽象类和抽象方法的特点和语法
  - Java基础类库的一些重要的类
  - JAR文件和jar命令
- 要求
  - 理解组合和继承的区别，能够知道何时使用那种方法
  - 了解终结类、终结方法、抽象类、抽象方法的概念
  - 熟练掌握本章提到的Java基础类库中的一些常见类
  - 初步了解JAR文件的概念，jar命令的格式