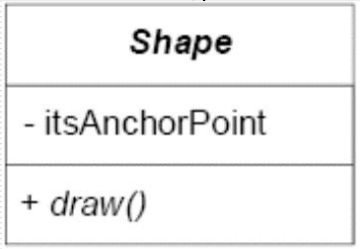
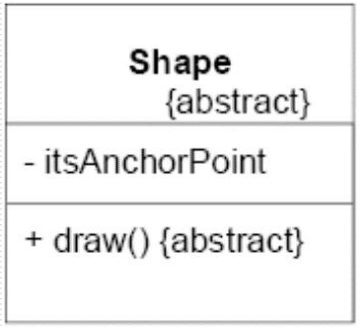
Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- ✓ 2.6 Abstract class
- 2.7 Interface

2.6 Abstract Classes (cont.)

- In UML there are two ways to denote that a class, or a method, is abstract.
  - You can write the name in italics,
  - or you can use the {abstract} property.






```
public abstract class Shape
{
  private Point itsAnchorPoint;
  public abstract void draw();
}
```

1. Abstract method

- A method is declared abstract by adding the keyword **abstract** to the method signature. for example

  ```
  public abstract void  sleep(int  hours);
  ```

- An *abstract method* consists of a method signature without a method body.
  - 函数的名称、参数列表、返回型别，但不含函数主体。

2. Abstract class

- An abstract method must be declared in an *abstract class*.
  - An abstract class is denoted by using the abstract modifier:
- public abstract class *className* {

  . . .

  }
- for example Container.java

2. Abstract class(cont.)

- abstract class 是一种对象类型，可以通过它来声明对象类型的变量。
  - Container container;
- No instance of an abstract class can be created.
  - container = new Container();//不可以

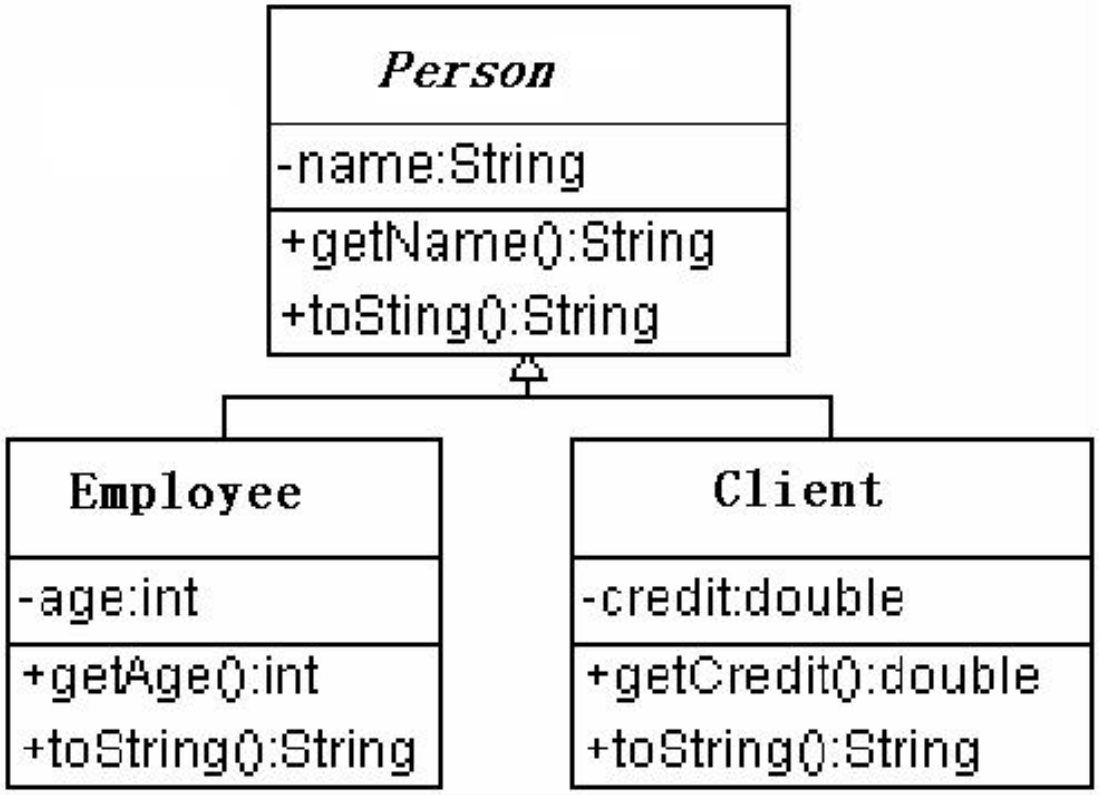
2. Abstract class(cont.)

- An abstract class may be extended to create subclass, in the same way that a 'normal' class can be extend.
  - If a subclass of an abstract class does not implement all abstract methods inherited from its parent, the subclass *must* also be defined as abstract.
- abstract class(抽象类) 是一种只可作为基类的对象类型。
  - 不允许通过new来调用其构造函数。

- **polymorphic variable**
  - **Container container**
    - **container = new Wagon(width, height, length);**
    - **container = new Tank((radius, length);**
- **polymorphic method**
  - **computeVolume()**

2. Abstract class(cont.)

- An abstract class **does not *have* to have abstract methods** (一个抽象类可以没有抽象方法) .
- For example:
  - Person.java
  - Client.java
  - Employee.java
  - PersonDemo.java



2. Abstract class(cont.)

- 抽象类与常规类的共同点与区别？
- abstract class存在的原因：
  - 让客户端程序员无法产生其对象，并因此确保这只是一个“接口”(而无实体)；
  - 共同的函数特征建立了一个基本形式，让程序员可以陈述所有子类的共同点。
    - 任何子类都可以以不同的方法体来表现此一共同的函数特征。
- 为图书馆系统创建的类图中的类CatalogItem可以是抽象的吗？
  - 进一步细化类图
- 其它？

1. Interfaces: “pure” abstract class.

- In Java, an interface defines a set of abstract methods. An interface is declared using the keyword interface.
  - public interface Device {

    …

    }
  - Like a class, you can add the public keyword before the interface keyword or leave it off to give “friendly” status so that it is only usable within the same package.
    - 和public类一样，public接口也必须定义在与接口同名的文件中。

1. Interfaces: “pure” abstract class. (cont.)

- All methods in an interface are implicitly public and abstract.
  - 无构造方法
- All data fields in an interface are implicitly public, static, and final.
- Interface让编程者得以创建出class的形式：函数的名称、参数列表、返回类型，但不含函数主体。
- Device.java

2. Implements an interfaces

- 接口是一种对象类型，可以声明该类型的变量，例如：Device device;
- For a class to make use of an interface, the keyword implements is used, followed by the name of the interface.
  - If the class does not implement all the methods, the class must be defined as abstract.
  - 所有实现interface的方法都必须被声明为public.
- 举例：The for-each loop can be used only on those classes that implement the interface java.lang.Iterable<T>.

2. Implements an interfaces(cont.)

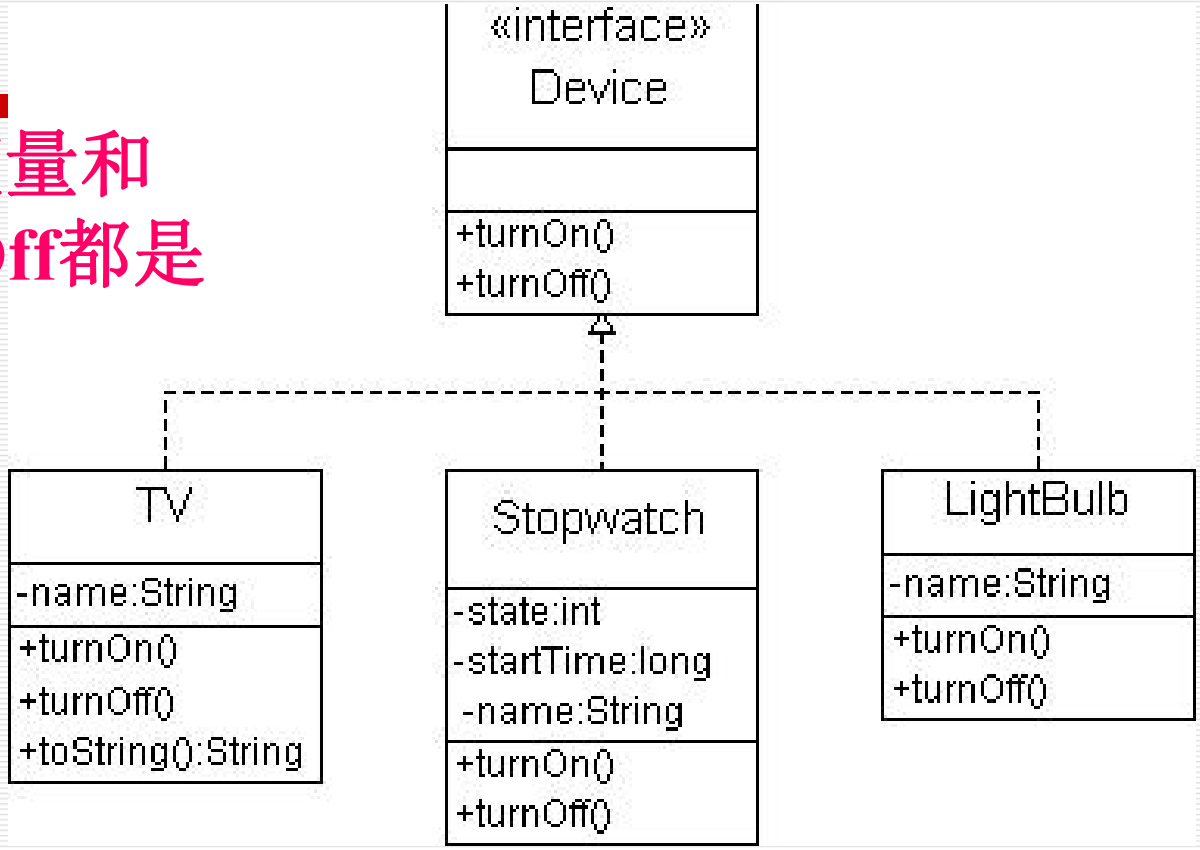
- For example _TV.java

```
public class TV  implements Device {
        ……
}
```

- 对于所有实现接口的类，其对象都可以赋给接口类型的变量。
  - Device device = new TV("TV");
- interface 作为基类的对象类型。
- 接口类型的变量和接口声明的方法都是polymorphic.

- 举例：

Device类型的变量和turnOn和turnOff都是多态的.



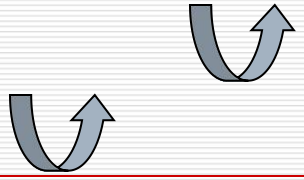
Device.java

TV.java

LightBulb.java

Stopwatch.java

DeviceDemo.java



3. Multiple inheritance in java

- A class can implement many interfaces.
- 一个类在继承另外一个类的同时，可以实现多个接口。
  - Adventure.java
- 使用接口，能够实现子类型被向上转型至多个基类型别；
  - 例如：Hero类型的对象可以被向上转型为ActionCharacter,CanFight, CanSwim或 CanFly 类型的对象。

3. Multiple inheritance in java(cont.)

- java不允许类的多继承，但允许接口的多继承。
- An interface can extend any number of interfaces.
  - 例如：

```
public interface MyInterface extends
        Interface1, Interface2 {
            void aMethod();
}
```

4. Comparing Abstract Classes and Interfaces

- An abstract class can define instance variables, whereas an interface cannot.
  - 举例：图书馆系统 CatalogItem
- An abstract class can implement methods, whereas an interface cannot.
  - 举例：图书馆系统 CatalogItem

4. Comparing Abstract Classes and Interfaces

- Interfaces do not contain constructors.
- If a class extends an abstract class, it cannot extend another class;
  - A class can only extend one class or an abstract class .
- If a class implements an interface, it is free to extend another class and implement many other interfaces.

4. Comparing Abstract Classes and Interfaces

- **Interface存在的原因：**
  - 使用接口，能够实现子型别被向上转型至多个基类型别(...)；
  - 让客户端程序员无法产生其对象，并因此确保这只是一个“接口”(而无实体)；
  - 共同的接口建立了一个基本形式，让程序员可以陈述所有的实现该接口的子型别的共同点。任何子型别都可以以不同的方法体来表现此一共同接口。

4. Comparing Abstract Classes and Interfaces

- 究竟应该使用interface还是使用abstract class？
  - Interface更优于abstract class，因为程序员能够藉以撰写出“可被向上转型为多个基类型别”的class，而达到c++多重继承的变形；
  - 如果撰写的base class可以不带任何函数定义或任何成员变量，应该优先考虑用interface；

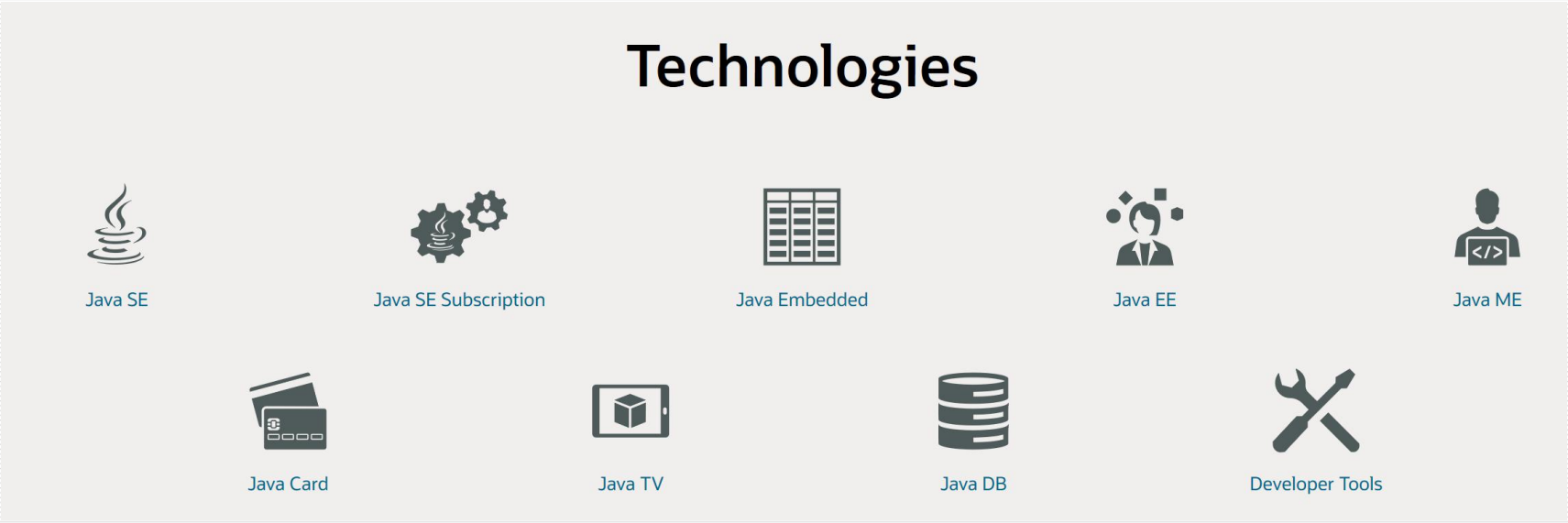
4. Comparing Abstract Classes and Interfaces

- 如果知道某个东西将会成为base class，你优先考虑使它成为interface，只有在必须带有函数定义或成员变量时，才使用abstract class，或必要时改为concrete class。
  - 为图书馆系统创建的类图中的类CatalogItem可以是建模为接口吗？
    - 进一步细化类图
  - 顾客信息系统
    - 进一步细化类图

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

1. Java 的开发工具包(Java Develop Kit)



https://www.oracle.com/java/technologies/javase-downloads.html

1. Java的开发工具包(Java Develop Kit)

- **J2SE(Java 2 Platform Standard Edition)（或Java SE） ---标准版(面向桌面应用开发)**
  - **是为Java应用程序, 提供Java标准运行环境支持的平台，包括输入\输出和图形用户界面等, 它是大多数开发人员应用标准JDK编程的Java语言平台。**
- **下载地址：**
  https://www.oracle.com/java/technologies/

1. Java 的开发工具包(Java Develop Kit)(续)

- **J2EE(Java 2 Platform, Enterprise Edition)(或Java EE)---企业版(面向企业级开发)**
  - **J2EE包含J2SE中的类，还包含应用于企业级应用的类，比如：EJB, servlet, JSP, XML等。**
    - **J2EE提供了一个基于组件的方法来设计、开发、装配和部署企业级应用程序的开发平台。 它将Java的企业级API捆绑在一起，成为Java语言进行服务器端的企业级应用部署的开发环境。**

1. Java 的开发工具包(Java Develop Kit)(续)

- **J2ME(Java 2 Platform, Micro Edition), Android Studio +Android APP**面向手机等移动终端开发
  - 包含**J2SE**中的一部分类。

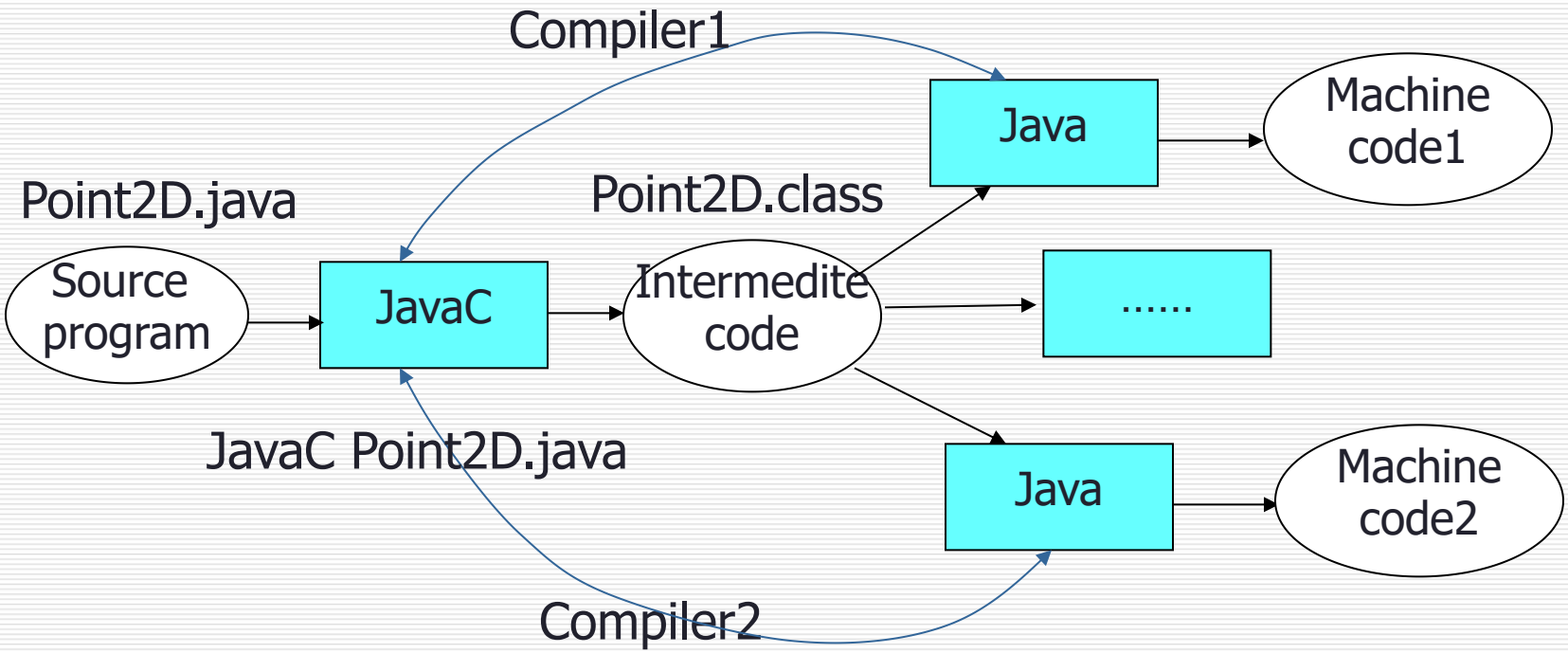
本课程的Java开发环境

- Eclipse集成开发环境：https://www.eclipse.org/downloads/
- 其它集成开发环境：Idea
- Use jdk1.8(Java Develop Kit)以上，阅读\ labwork \ readme.txt文件，熟练掌握jdk各命令：
  - javac
  - java
  - javadoc
  - jar
  - UltraEdit-32或Notepad++:一套极棒的文字、Hex、ASCII码编辑器，可以取代记事本来编辑java源文件。

Java项目开发

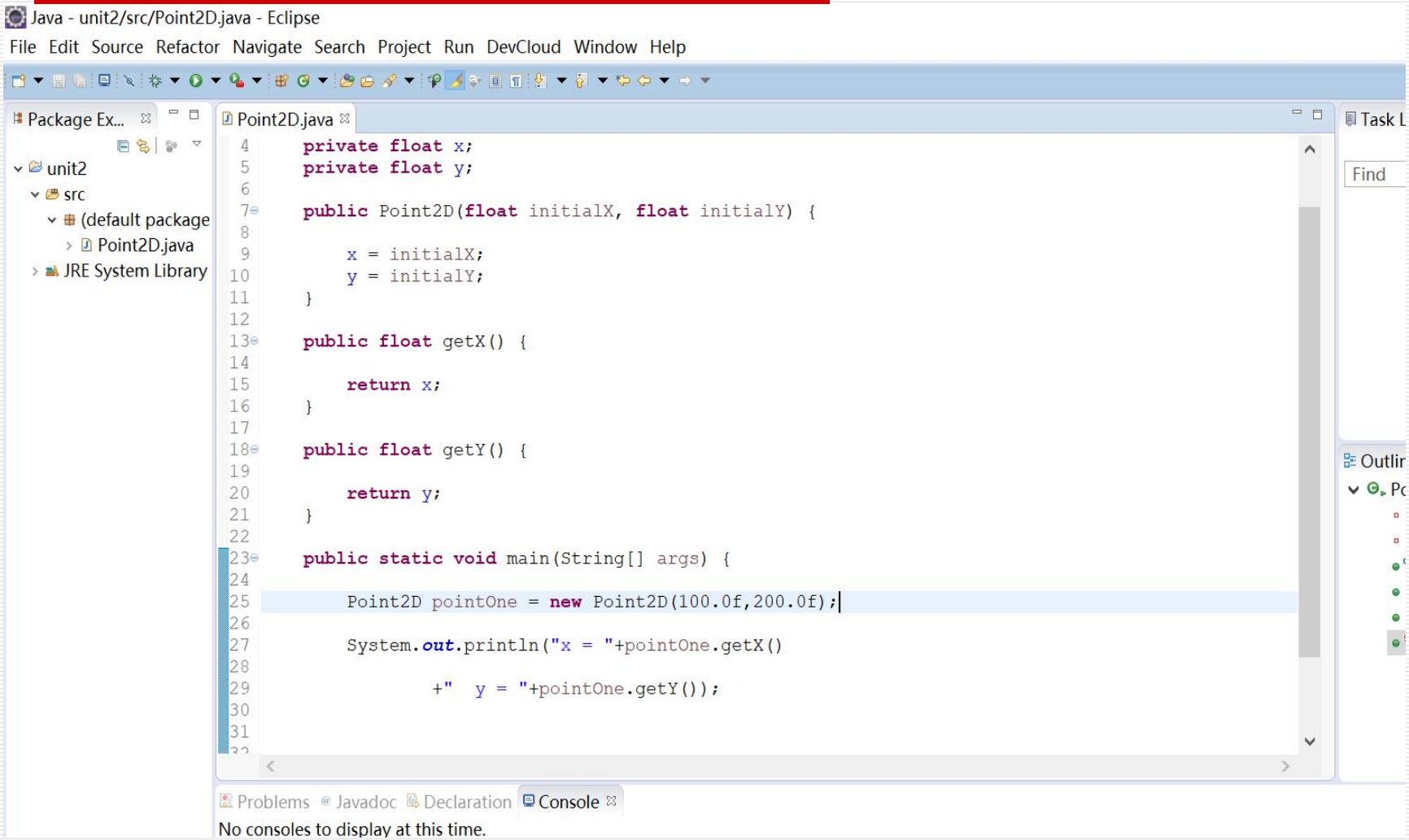
- Java：
  - 支持**Web软件**开发。以java的Spring框架为后端，借助前端技术开发界面，并使用electron把前端资源打包为桌面程序
  - 支持**桌面版 （单机版）**软件开发。以idea为集成开发平台，使用JavaFX框架开发java原生界面程（JavaFX支持拖拽方式生成界面代码）
  - 支持**eclipse插件开发**。eclipse IDE为插件开发平台，以JDT和PDE（Eclipse SDK）为主要开发环境进行插件开发，OSGi框架、RCP框架。
  - 支持**idea插件开发**。以idea IDE为插件开发平台，以InteliJ SDK为主要开发环境，以Swing为组件界面开发库进行插件开发，并使用Gradle进行插件打包发布。
  - 支持**AI相关的算法任务构建**。Java机器学习相关的库有Java-ML，RapidMiner和Weka（同时提供GUI平台和Java API）。Java深度学习相关的库有Neuroph，Deeplearning4j，以及OpenNLP，CoreNLP等等。

Hello Java



- JavaC:将Point2D.java源程序翻译为Point2D.class
- Java:在不同的计算平台上解释运行Point2D.class

Use eclipse(java集成开发平台)



Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

2.1.1 Java class and object

- 今日向题簿：

1. 撰写java类的语法

2. Class

- **Creating new data types: class**
  - The **class** keyword is followed by the name of the new type. For example:

```
public class Name {
}
```

```
class Name {
}
```

- **When you define a class, you can put data members (*fields/state/attributes*) and member functions (*methods/operations/behavior*) in your class.**

2. Class (cont.)

- Data members (***fields/state/attributes***)
  - **A data member is an object of any type that you can communicate with via its reference. It can also be one of the primitive types.**
    - 访问权限 数据类型 变量名
      - 四种访问权限：public,private,protected,缺省
      - Java的数据类型包括：基本数据类型和对象类型
      - DNASequence.java Triangle.java

2. Class (cont.)

- Java的数据类型

2. Class (cont.)

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	1-bit	–	–	**Boolean**
char	16-bit	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8-bit	-128	+127	**Byte**[1]
short	16-bit	-2^{15}	$+2^{15}-1$	**Short**[1]
int	32-bit	-2^{31}	$+2^{31}-1$	**Integer**
long	64-bit	-2^{63}	$+2^{63}-1$	**Long**
float	32-bit	IEEE754	IEEE754	**Float**
double	64-bit	IEEE754	IEEE754	**Double**
void	–	–	–	**Void**[1]

All numeric types are signed, so don't go looking for unsigned types.

BigDecimal
BigInteger

2. Class (cont.)

Member functions(*methods/Operations)*

- **The fundamental parts of a method are the name, the arguments, the return type, and the body. the basic form:**

```
returnType methodName( /* argument list */ ){
    /* Method body */
}
```

- 参数列表可以为空，参数列表的格式如下：

参数类型1 参数名1，参数类型2 参数名2，……

- **Java中的方法只能作为类的一部分创建（与C++不同）**

2. Class (cont.)

- 方法体的撰写
  - 与C语言相同，课下复习、查阅；
    - 循环、分支等流程控制语句
    - 赋值、变量的作用域规则
    - 运算符等
    - 举例：DNASequence.java

2. Class (cont.)

- Method overloading(方法的重载) allows two or more different methods belonging to the same class to have the same name as long as they have different argument signatures（the argument types and their order）

```
public class Calculator {
        …
        public static int add(int x,int y) {
                reutrn x+y;
        }
        public static double add(double x,double y)  {
                return x+y;
        }
}
```

2. Class (cont.)

- Java类的构造方法
  - 构造方法的名字和类名相同，并且没有返回值(注：其它语法要求与普通方法一样)；
    - use the prefix "initial" to name the parameters of constructors
  - 构造方法的作用:
    - 构造方法通过传递过来的实参初始化对象的成员变量；
    - new关键字调用构造方法来创建类对象；

2. Class (cont.)

- Java类都要求有构造方法，如果没有定义构造方法，java编译器会提供一个缺省的构造方法，即不带参数的构造方法。缺省构造方法用默认值初始化对象的成员变量，各种数据类型的默认值为：
  - 数值型(基本数据类型) 0
  - boolean false
  - char '\0'
  - 对象类型 null

2. Class (cont.)

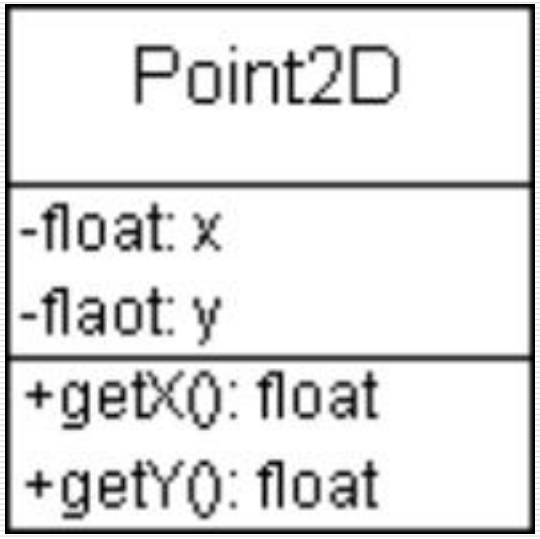
- Constructors can be overloaded. Constructors can call other constructors (unlike C++).
  - 当你为某个类撰写多个构造函数时，想要在某个构造函数里调用另一个构造函数，用this([参数列表])
  - Overloading.java

2. Class(cont.)

- When you create a source-code file for Java, it's commonly called a *compilation unit*. each compilation unit must have a name ending in .java.
- There can be only *one* public class in each compilation unit. Inside the compilation unit there can be a public class that must have the same name as the file.
  - SprinklerSystem.java

2. Class (cont.)

- 举例：写出类Point2D的源文件Point2D.java，要求类Point2D提供一个有参构造函数，对属性x，y进行初始化。



类Point2D

Java代码结构

Point2D.java

```
{…
  private float x;
  private float y;
  public Point2D(float initialX, float initialY) {
                x = initialX;
                y = initialY;
  }
  public float getX() {
                return x;
  }
  public float getY() {
                return y;
  }
  public static void main(String[] args) {
                        //your code here
   }
…
}
```

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

2.1.1 Java class and object

- 今日问题簿：

1. Java对象如何创建与使用？
2. 何为Java引用？

3.Object Initialization and Application

- The term instantiation is used to refer to the process by which an object is created/constructed based upon a class definition.
  - 在面向对象系统中，常常把对象和实例当作同义词。
- 在类定义的基础上，其对象的创建与使用通常包括三个步骤：
  - 1）对象变量（或引用）的声明
  - 2）对象的创建
  - 3）对象的使用

3.Object Initialization and Application(cont.)

- 1）对象变量的声明：In java, when we declare a variable to be of a user-defined type, like:

  Point2D pointOne;(compared with C++)

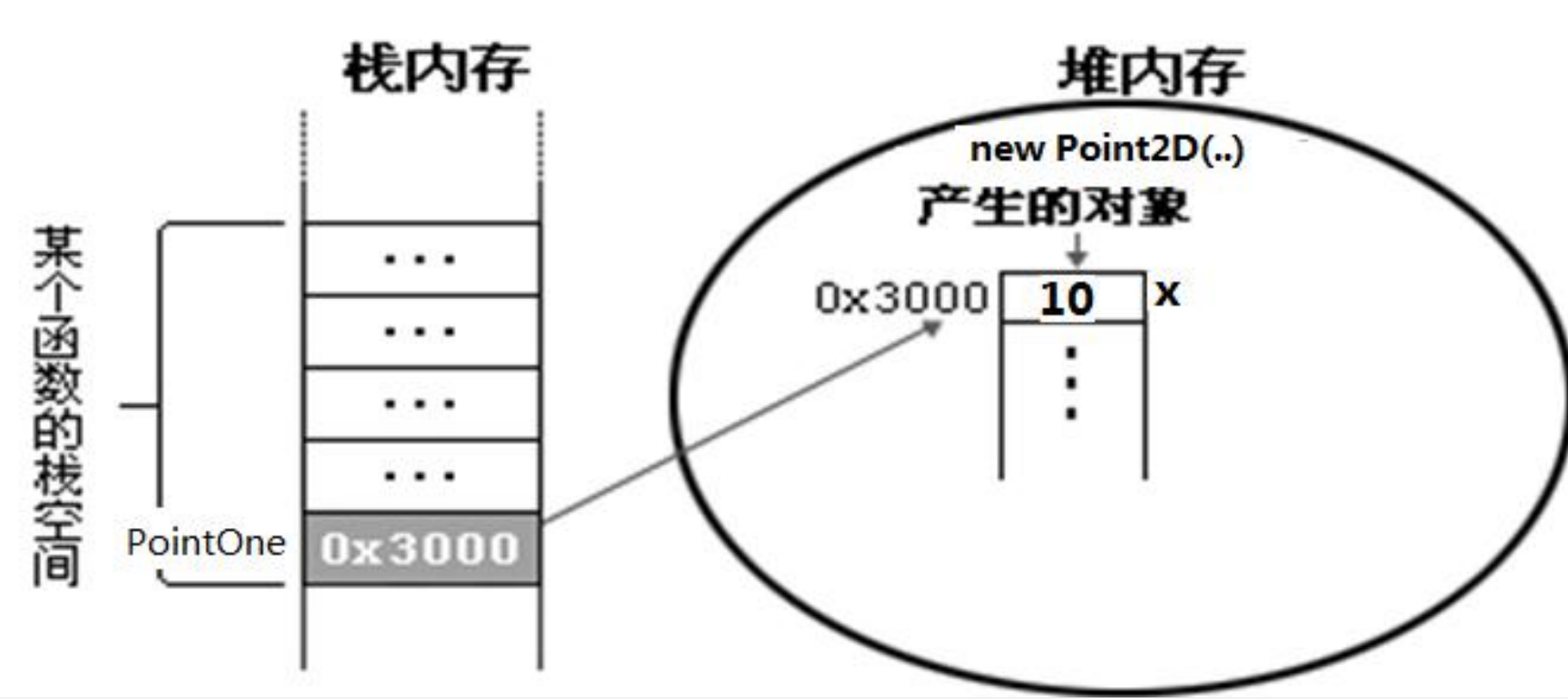
  - We haven't actually created an object in memory yet.
  - 对于对象变量pointOne的声明，操作系统会在栈(Stack)内为pointOne分配一个存储地址数据的内存空间（32位）；
    - 该内存空间将存储某个Point2D对象内存空间的首地址；

3. Object Initialization and Application(cont.)

- 2）对象的创建：The new operator, to actually carve out a brand new object in memory, as follows:

  pointOne = new Point2D(10, 100);

  - 使用new调用类的构造函数创建对象，操作系统会在堆(Heap)中分配对象所占的存储空间。
- pointOne存储对象new Point2D(10, 100)分配的内存空间首地址。所以pointOne就是一个引用（references），其值指向对象所在内存中的位置。

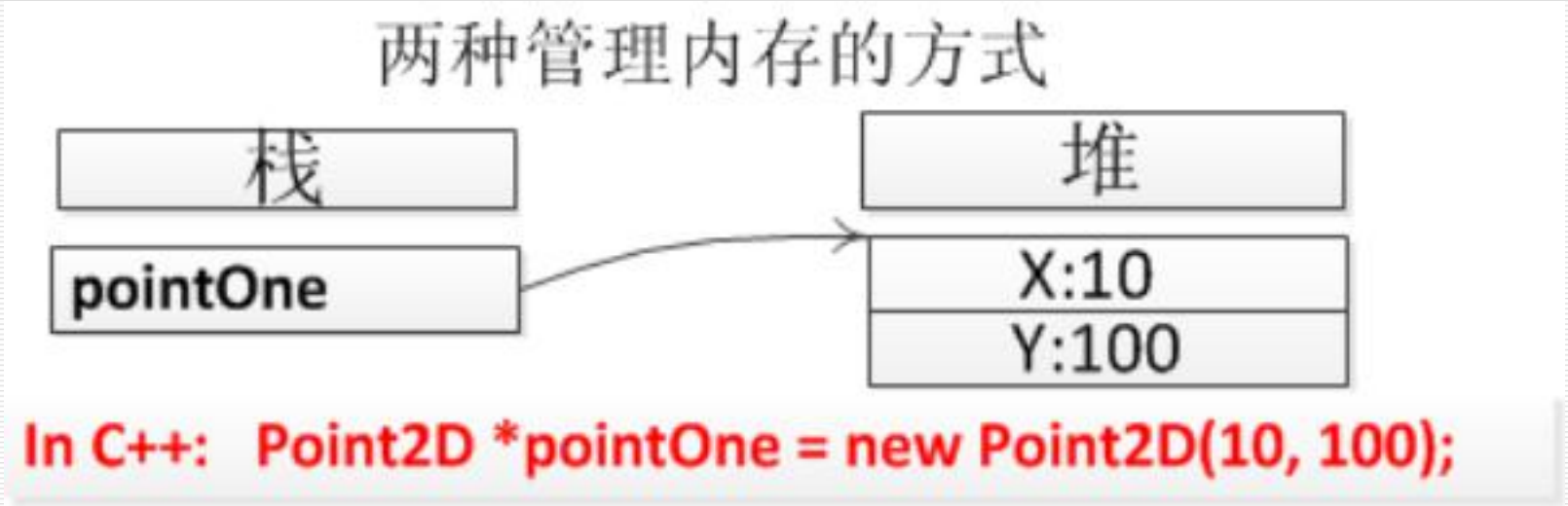

栈内存
堆内存
new Point2D(..)
产生的对象
某个函数的栈空间
PointOne
0x3000
0x3000
10
x

3. Object Initialization and Application(cont.)

- **New关键字的每次使用，会创建相应类的一个新的对象，一个类的不同对象分别占据不同的内存空间。New关键字的作用如下：**
  - （1）引起对象构造方法的调用。
  - （2）为对象分配内存空间。
  - （3）为对象返回一个引用。
- **对象的清除**
  - 当对一个对象的引用不存在时，该对象成为一个无用对象。Java的垃圾收集器自动扫描对象的动态内存区，把没有引用的对象作为垃圾收集起来并释放。

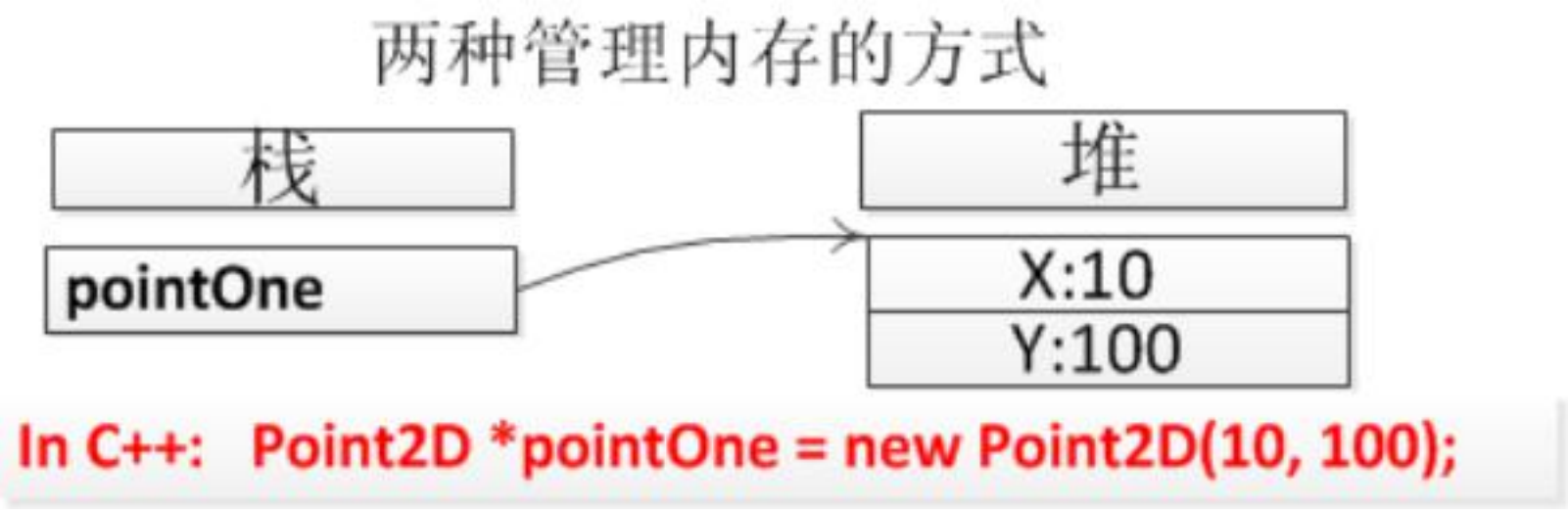
3. Object Initialization and Application(cont.)

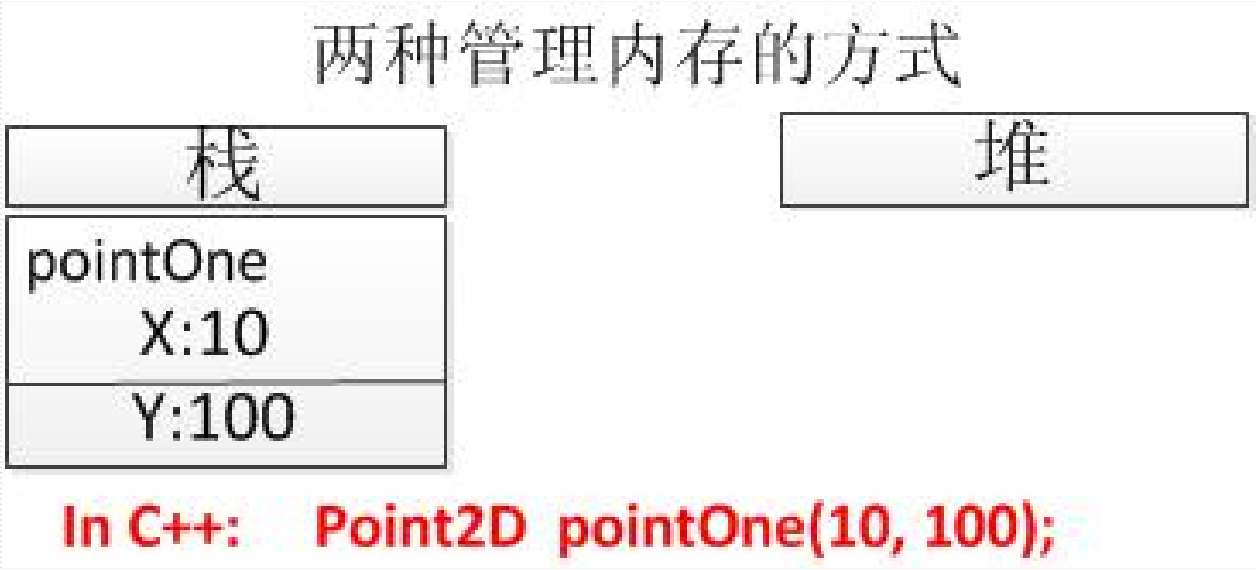
In java: Point2D pointOne = new Point2D(10, 100);



In C++: Point2D *pointOne = new Point2D(10, 100);

Java compared with C++





3. Object Initialization and Application(cont.)

- 3）对象的使用：通过运算符“.”可以实现对对象属性的访问和方法的调用，访问格式如下：
  - reference.variable 和 reference.methodName([paramlist])
- 上述reference可以是已初始化的对象变量，也可以是生成对象的表达式，例如：
  - float tx =pointOne.x;
  - float tx = new Point2D(100.0, 200.0).x;
  - pointOne.setX(300.0);
  - new Point2D(100.0,200.0) .getX( );
    - 举例：Point2D.java　　Triangle.java

3. Object Initialization and Application(cont.)

- 有一个现成的类，如何用它来干活（实现功能）？
  - 1）？
  - 2）？
- Java方法调用中形参和实参的结合方式：
  - 2016 object-oriented design and programming\unit2 java Class Implementation\java-source\2.1.1\ Java形参与实参的结合方式演示

4. 对象生命周期

1. 类型为Point2D的一个对象首次创建时，或者Point2D类的静态方法或数据首次调用时，java解释器必须查找环境变量classpath所指定的位置，找出并装载Point2D.class。
2. 一旦Point2D.class装载后，它的所有静态初始化模块都会运行，静态初始化动作仅发生一次，就是在Point2D.class首次被装载时。

4. 对象的生命周期(续)

3. 如果1. 中所指是创建一个Point2D类型的对象(例如new Point2D(10,100))，则顺序执行以下步骤：

   3.1 那么new语句首先会在堆里分配一个足够的存储空间。

   3.2 系统会先用默认值初始化对象的成员变量，因此Point2D中的所有数据成员都得到了缺省值。

   3.3 执行所有出现于数据定义处的初始化动作。

   3.4 执行构造函数。

✓ 如果应用程序没有结束，在同一个应用程序中再创建Point2D类型的对象，则从步骤3中的3.1开始顺序执行。

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

2.1.2 Java access control

- Java对象是Java封装性的一个方面，封装性的另一方面体现为对Java 类和其成员的访问权限限制。Java的访问权限限制包括包、类的访问权限以及类成员的访问权限三个部分 -------使用Java类完成功能必备的基础知识。

1. Java包

- 类命名冲突问题？
  - Java引入包(package)来组织、管理类，解决类命名冲突的问题；
- 包存放一个或多个相关类的集合,对应于文件系统的目录层次结构，它通过“.”来指明目录的层次，例如：包名=java.io，其对应的文件系统的目录结构为： …\java\io
  - 类位于包中，即类位于相应的文件夹中
  - The classes in the Java API（Application Programming Interfac） are grouped into packages;

1. Java包(续)

- 例如：包java.util对应的目录结构？
  - 类Date位于java.util包中，它在哪个目录下？
  - 查看java类库，理解包的概念 …\Java\jre1.5.0\lib\rt.jar
- Java包 (package)中存放的是一个或多个相关类的集合，例如：
  - 所有与输入和输出相关的类都放在java.io包中，与网络功能有关的类都放到java.net包中。

1. Java包(续)

- 类的使用：
  - 1.用类的全名= 包名+类名
  - 2. the import Statement
- To declare a variable of type Date,we would type:
  - 其中，Date是类的名字，java.util是类Date所属的包的名字，java.util.Date是类的全名；

```
java.util.Date day =  new java.util.Date();
```

  - 通过这种方式来唯一地标示一个类名， 所以，包提供了一种命名机制和可见性限制机制。

1. Java包(续)

The import Statement

- Java offers the import statement. It is used to "import" a class—or an entire package of classes—into a file.

```
import  java.util.Date
//This makes it possible to write:
Date day = new Date();
```

- 在同一个源文件中，还使用同一个包中的其它类，怎么导入？

1. Java包(续)

- Often, programmers use several classes from the same package. the entire package is imported:

```
import  java.util.*
```

This makes it possible to write:

```
Date day = new Date();
Dictionary dictionary = new Dictionary();
```

1. Java包(续)

定义包

- The packege keyword, at the beginning of a file.(package语句必须是文件中的第一条语句)

```
packege  point;
public class Point2D {
    ……
}
```

1. Java包(续)

- If anyone wants to use the name they must either fully specify the name or use the import keyword, For example point/Point2D.java, Triangle.java

① point.Point2D pointOne = new point.Point2D(1，2);

② import point.*;

…

Point2D pointOne = new Point2D(1，2); or

Point2D pointOne;

1. Java包(续)

- 如果一个.java源文件中不加package语句，则指定为缺省包或无名包，类文件位于当前工作目录下。
- 无论是jdk提供的类还是自己定义的包中的类，都必须用import语句标识或使用类的全名，以通知编译器在编译时找到相应的类文件，下述两种情况除外。
  a) 位于同一个包内的类可以相互引用，不必使用import语句或类的全名；
  b) 在源程序中用到了java类库中java.lang包中的类，可以直接引用，不必使用import语句或类的全名。

2.1.2 Java access control

2. Class access

- To control the access of a class, the specifier must appear before the keyword class.
- You have only two choices for class access: public or default (friendly)
  - 缺省情况下类的访问权限为“friendly”，只有同一个包内的其它类可以访问。
  - public类可以被任意包中的任意类访问。
    - 举例： point/Point2D.java, Triangle.java

3. Java access specifiers

- The Java access specifiers public, protected, and private are placed in front of each definition for each member in your class, whether it's a field or a method.
- The default access has no keyword, but it is commonly referred to as "friendly."

3. Java access specifiers(cont.)

- The private keyword means
  - 除了该私有成员所在的类，没有任何类可以访问/使用这个成员，即使是同一个package内的其它类, 也无法访问/使用该类的private成员。
- “friendly” means
  - 与friendly成员所在的类位于同一个package的所有类可以访问friendly成员，但是该包之外的任何类都不可以访问friendly成员。
- 举例： point/Point2D.java, Triangle.java

3. Java access specifiers(cont.)

- **protected: "sort of friendly",**

  何时用到该访问权限？

  - **与protected成员所在的类位于同一个package内的类都可以访问protected成员；但是在该包之外，除了其子类，其它任何类都不可以访问protected成员。**
    - **基类中的protected成员允许其子类访问，无论子类是否与基类位于同一个包。**
- **public:**
  - **意味着该成员可以被任意包中的任意类访问**
    - **in particular to the client programmer who uses the library.**

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

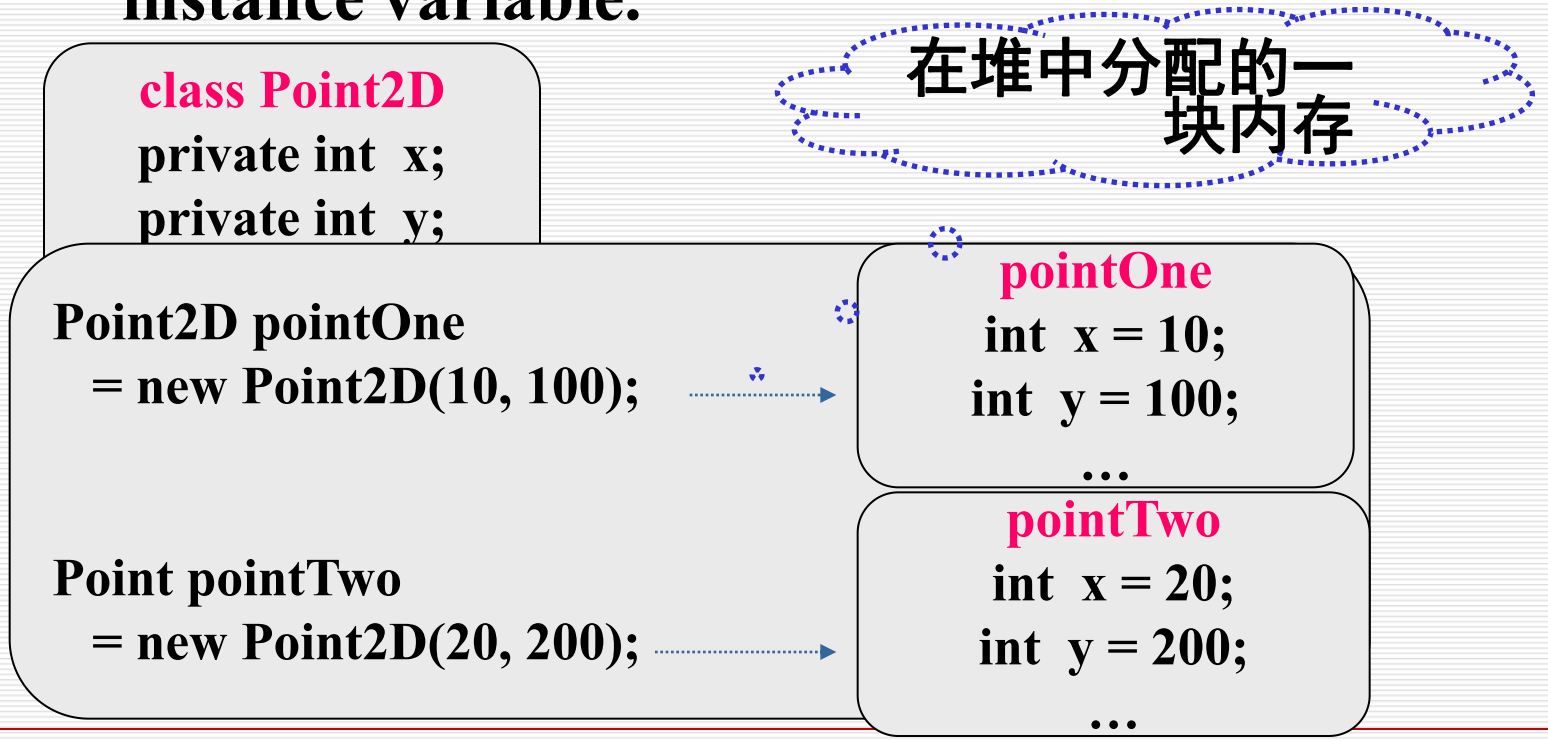
2.1.1 Java class and object

- 今日问题簿：

1. 是否理解实例变量、静态变量、实例方法和静态方法？

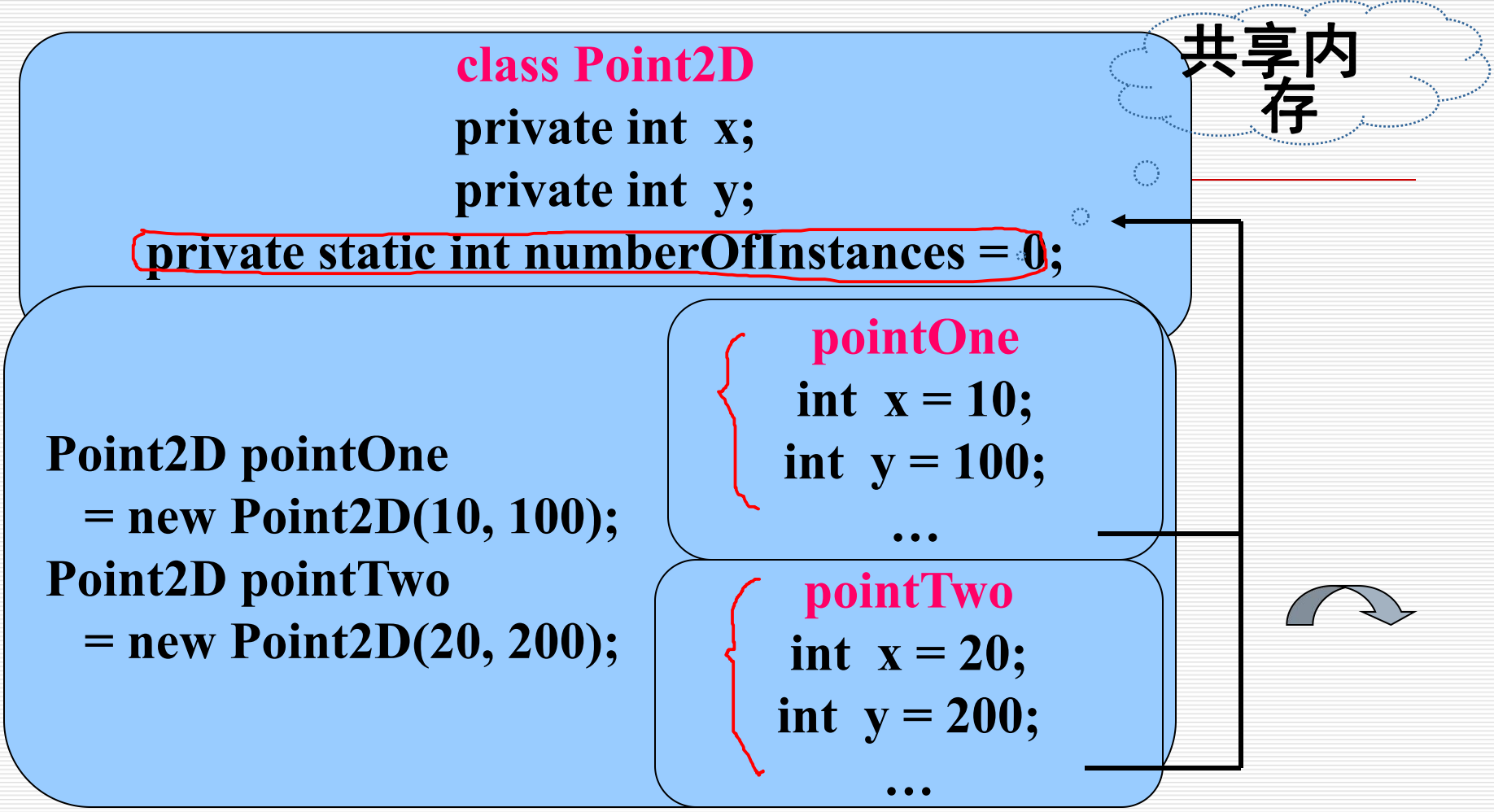
4. Instance/Static Variables and Methods

- instance variable:没有加static关键字的变量称为类的实例变量成员。
  - each object of the class will have its *own* copy of the instance variable.



4. Instance/Static Variables and Methods (cont.)

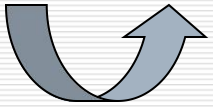
- class variable (*static variable*):
  - 加了static 关键字的成员变量称为静态变量或类变量。
    - A static attribute is one whose value is shared by all objects of that class;
      - A static attribute belongs to the class as a whole instead of belonging to any one instance/object of that class.
- 有时候需要创建类的变量，而不是对象的变量，例如：统计点的数量(要求记录Point2D类到目前为止共创建了多少个Point2D类对象);
- For example Point1/Point2D.java



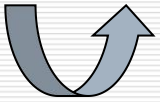
- 类变量位于类所有对象共享的内存空间；

4. Instance/Static Variables and Methods (cont.)

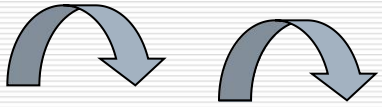
- **instance method** :类内没有加static关键字的方法称为类的实例方法。
  - **can access *both* instance and class variables/methods.**
  - 一个类所有的对象调用的成员方法只有一份(…)。
  - **Point2D pointOne = new Point2D(10, 100);**
    pointOne.setX(30);
    Point2D pointTwo = new Point2D(20, 200);
    pointTwo.setX(50);
    实例方法怎么与对象关联?



4. Instance/Static Variables and Methods (cont.)

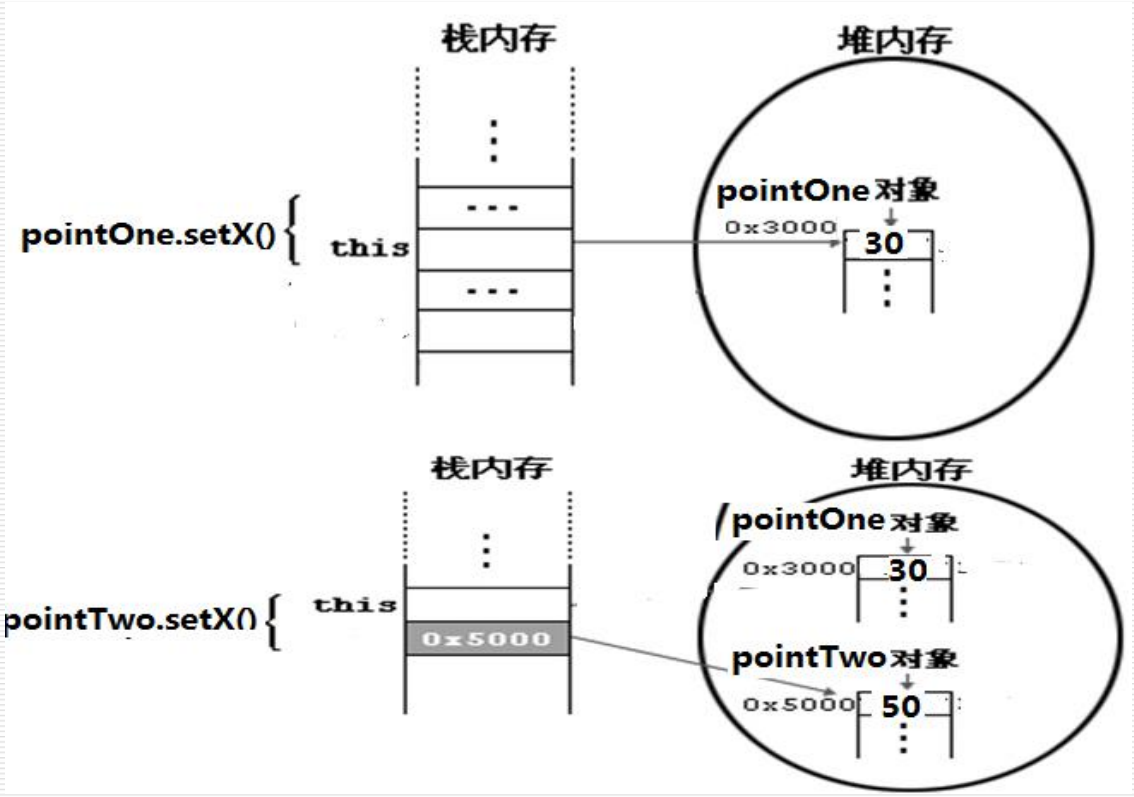


- public void setX(int newX) {
  x = newX;
  }
  - 被编译器翻译为下述函数：
    public void setX(Point2D this, int newX) {
    this.x = newX;
    }
- pointOne.setX(30);
  - 翻译为：setX(pointOne , 30);
- pointTwo.setX(50);
  - 翻译为：setX(pointTwo , 50);



4. Instance/Static Variables and Methods (cont.)

- 每个实例成员方法内部，都有一个this变量，指向调用该方法的具体对象。
  - this变量允许相同的类实例方法为不同的对象工作。



4. Instance/Static Variables and Methods (cont.)

- **class method (*static method*):**
  - 类内加了static关键字的方法称为静态方法或类方法；
    (注：构造方法前不允许加static关键字)
    - 类的静态方法只访问该类的静态变量；
    - 类的静态方法只能调用该类的静态方法；
- **For example**
  - **静态方法main(),每个类都可以有main()方法**
    - **Point1/Point2D.java**
    - **SprinklerSystem.java**

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

1.1.3 Beginning with the Java API

- Java API应用举例
- 今日问题簿：
  - 会使用String类处理字符串吗?
  - 控制台读写如何编程?
  - 会使用StringTokennizer类完成字符串解析的任务吗?

1. java.lang.String Class

- String:Java has ‘real’ character strings,as opposed to the use of char(acter) arrays to simulate strings in C .
- 在Java中，存储和处理字符串等相关功能的实现，使用String类。

1. java.lang.String Class

The java.lang.String Class

- ***String(String original)***. Constructs a new String object that represents the same sequence of characters as the argument.
- ***int length()***. Obtains the number of characters in the String.
- ***char charAt(int index)***. Returns the character at the specified index.
- ***boolean equals(Object anObject)***. Returns true if the specified Object represents a String with the same sequence of characters.
- ***int indexOf(int ch)***. Returns the index of the first occurrence of the character.
- ***int indexOf(String str)***. Returns the index of the first occurrence of the String.
- ……

1. java.lang.String Class

- 类String的equals( )方法，比较两个字符串的内容是否相等。
- 用连接号“+”或 类String的静态方法valueOf( )可以将数值转化为字符串
- 特别注意：
  - == compares the addresses of the objects, not the objects themselves.
  - ==可以比较基本数据类型的数值
    - double类型的值比较.txt
- StringClassDemo.java DNASequence.java

1. java.lang.String Class

- Java provides the String concatenation operator (+);
  - "hello " + anyObject.toString() ;
  - "hello " + anyObject
- 转义字符：
  - To insert a tab character in a String literal,use \t
  - To insert a hard return in the middle of a String literal,use \n or \r\n
  - To insert a double quote in the middle of a String literal,use \”
  - To insert a backslash in the middle of a String literal,use \\

2 Console I/O(The java.io Package)

Reading from standard input

```
BufferedReader stdIn =
        new BufferedReader(new
                InputStreamReader(System.in))
            ……
String input = stdIn.readLine();
```

- stdIn.readLine();返回从控制台读取的字符串类型的对象

2 Console I/O(The java.io Package cont.)

Standard output

- PrintWriter has a constructor that takes an OutputStream as an argument.

```
PrintWriter stdOut = new PrintWriter(System.out, true);
```

- stdOut is used to display typical and regular program output.

```
stdOut.println("the result is :");
```

2 Console I/O(The java.io Package cont.)

Error output

- System.err is the "error" output stream. Typically, this stream corresponds to screen output.

```
PrintWriter stdErr =new PrintWriter(System.err, true);
```

- stdErr is used to display prompts and error messages.

```
stdErr.println("Incorrect number format");
```

- Hello.java

1.1.3 Beginning with the Java API

- Java API应用举例
- 今日问题簿：

3. The java.util.StringTokenizer Class

- ***Tokenizing*** **is the process of breaking a string into smaller pieces called *tokens*.**
  - **Tokens are separated, or delimited, by a character or group of characters.**
- **StringTokenizer一次可以返回字符串内的一个词汇单元，字符串被分割符划分为若干词汇单元。**
  - **分割符可以采用默认或用户指定的形式。**

3. The java.util.StringTokenizer Class

- 例如：
  - This string has five tokens
    - 该字符串以空格为分割符将会被StringTokenizer划分为5个词汇单元。
  - Mini Discs 74 Minute (10-Pack)_5_9.00
    - 该字符串以下划线“_”为分割符将会被StringTokenizer划分为3个词汇单元。
- 试想该类应该提供什么样的操作完成词汇单元分割的功能？

3. The java.util.StringTokenizer Class

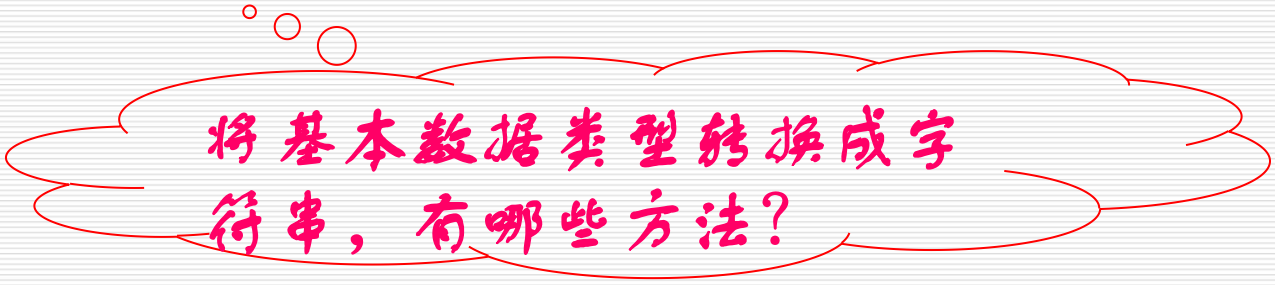
- public StringTokenizer(String str)
  - Str是用户指定的要分割的字符串
  - 默认的分割符：the space character, the tab character, the newline character, the carriage-return character, and the form-feed character
- public StringTokenizer(String str, String delim)
  - Str是用户指定的要分割的字符串
  - Delim是用户指定的分割符

3. The java.util.StringTokenizer Class

- public String nextToken()
  - Returns the next token from this string, The current position is advanced beyond the recognized token.
- public boolean hasMoreTokens()
  - Tests if there are more tokens available from this tokenizer‘s string. The current position is not advanced.
- public int countTokens()
  - Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception. The current position is not advanced.
- StringTokenizerClassDemo.java

4. The Wrapper Classes (cont.)

- Byte、Short、Integer、Long、Character、Float、Double、Boolean
  - The wrapper classes provide methods for converting primitives to String objects and String objects to primitives.
  - For example
    - Integer.toString(10); Integer.parseInt("10");



4. The Wrapper Classes (cont.)

- 功能要求：
  - 已知一个SaleItem类
  - 试撰写一个类Generator，该类包含一个静态方法：
    public static SaleItem createSaleItem(String str, String deli)
    - 该方法将包含销售项信息的字符串str以deli作为分割符将其划分为相应的销售项属性信息，并创建一个SaleItem对象返回。
      - Generator.java

4 Console I/O(The java.io Package cont.)

Reading Primitive Values

- 功能要求：
  1. 提示用户由控制台输入三个整数，以空格区分；
  2. 从控制台读取以空格区分的三个整数；
  3. 然后将三个整数相加
  4. 将相加的结果打印到控制台上。
- for example ReadThreeIntegers.java

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

2.1.4 Exception Objects

异常的引入

1. Exception Handling
2. Throw
3. The try-catch-block
4. Exception Classes and throws
5. Exception object information
6. finally
7. User-Defined Exception

要求掌握

今日问题簿：

哪几个关键构成了Java异常处理机制的基本框架？各关键字的用法？

用户自定义异常(了解)

1. Exception Handling

- 程序运行异常：指由程序运行环境问题造成的程序异常终止,包括设计者和编程人员的疏忽、使用者的误操作、软硬件错误等。
  - 为了保证程序的健壮性(robust)，必须在程序中对它们进行预见性处理。

```
int readInteger () {
        Read a string from the standard input
        Convert the string to an integer value
        Return the integer value
    }
```

1. Exception Handling(cont.)

- In traditional programming, To create a robust method, you must include conditional statements to detect and handle program failures.

```
int readInteger () {
    while (true) {
        read a string from the standard input;
        if (read from the standard input fails) {
            handle standard input error;
        } else {
            convert the string to an integer value;
            if (the string does not contain an integer) {
                handle invalid number format error;
            } else {
                return the integer value;
            }
        }
    }
}
```

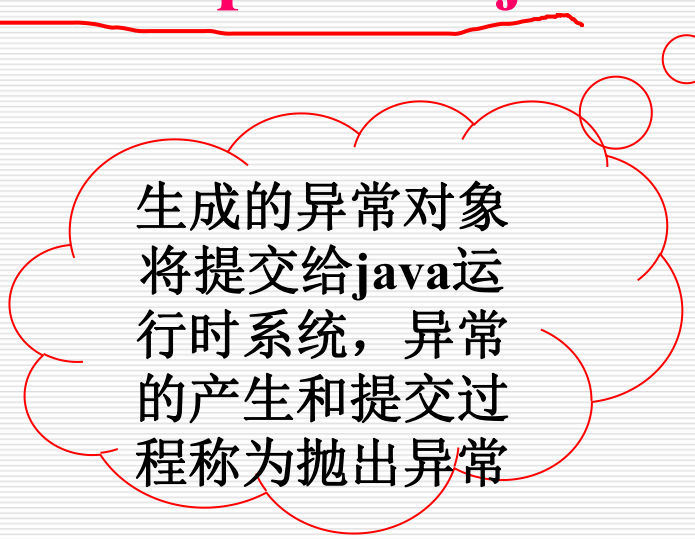
1. Exception Handling(cont.)

- ***Exception handling in Java*** **is a mechanism that allo nor**
  - **Th** **y to m**

```
int readInteger () {
    while (true) {
        try {
            read a string from the standard input;
            convert the string to an integer value;
            return the integer value;
        } catch (read from the standard input failed) {
            handle standard input error;
        } catch (the string does not contain an integer) {
            handle invalid number format error;
        }
    }
}
```

2. throw

- In Java, an exception is an object that describes an abnormal situation.
- When an abnormal situation occurs, a method can create an exception object and then throw it.



2. throw(cont.)

- Create an exception object and throw
  - throw new Exception("This is an Exception Message");
  - throw new Exception( );
  - Throw出现在方法体中，用于抛出异常，即当方法在执行过程中遇到异常情况时，将异常信息封装为异常对象，然后抛出。
    - 抛出异常意味着通知应用程序发生了错误，目的是为了给应用程序提供从问题中恢复和处理的机会。
- *SquareRoot.java*

3. The try-catch Block

- If this method choices to handle an exception, it must have a try-catch block.
  - *SquareRoot.java*
  - ExceptionDemo.java
- A try block is followed by one or more catch blocks.

3. The try-catch Block(cont.)

```
Try {
    //Code that might
    generate exceptions
} catch (Type1 id1) {
    //handle exceptions of
     Type1
} catch (Type2 id2) {
    //handle exceptions of
     Type2
}//etc…
```

- *SquareRoot.java*

- Try出现在方法体中，它包含一个作用域，表示尝试执行该作用域内的语句，在执行过程中可能有某些语句抛出异常；
- Catch出现在try之后，用于捕获和处理try中可能抛出的异常。Catch关键字后面紧跟着它能捕获的异常类型，它也包含一个作用域，作用域内的语句是对异常的处理。

3. The try-catch Block(cont.)

- 如果try代码块中的代码在执行时没有发生异常，所有的catch代码块会被跳过，并继续执行最后一个catch代码块之后的程序代码；
- 如果执行try代码块时方法抛出一个异常，try代码块的执行在产生异常的那行代码中止，然后JVM从上到下检查catch关键字后声明的异常类型和抛出的异常类型匹配的catch子句(找到？找不到？)。

3. The try-catch Block(cont.)

- JVM查找异常类型匹配的catch子句：
  1. 如果找到匹配的catch子句，JVM执行相关的catch代码块。在多个catch子句匹配的情况下，只执行第一个匹配的catch代码块，然后继续执行最后一个catch代码块之后的程序。
     - For example ***SquareRoot.java***

3. The try-catch Block(cont.)

2. 如果这个方法没有捕捉这个异常对象(JVM没有找到匹配的catch代码块或没有相应的try-catch块)：
   - 它就将异常对象掷出给调用它的方法，依此类推，直到被捕捉。如果被调用的方法都没有捕捉该异常对象，异常可能一路往外传递直达main()而未被捕捉，则运行时系统将终止，相应的java程序也将退出，最后会在命令行窗口报告抛出的异常信息。
   - for example ExceptionDemo.java

- 总结：写类的方法体的时候，是否有异常发生，你如何处理？

4. Exception Classes and throws

- Exception objects are instances of classes that descend from class Throwable. Throwable has two subclasses: Exception and Error.
- Exception is the basic type that can be thrown from any of the standard Java library class methods and from your methods and Java runtime accidents. So the Java programmer's base type of interest is Exception.

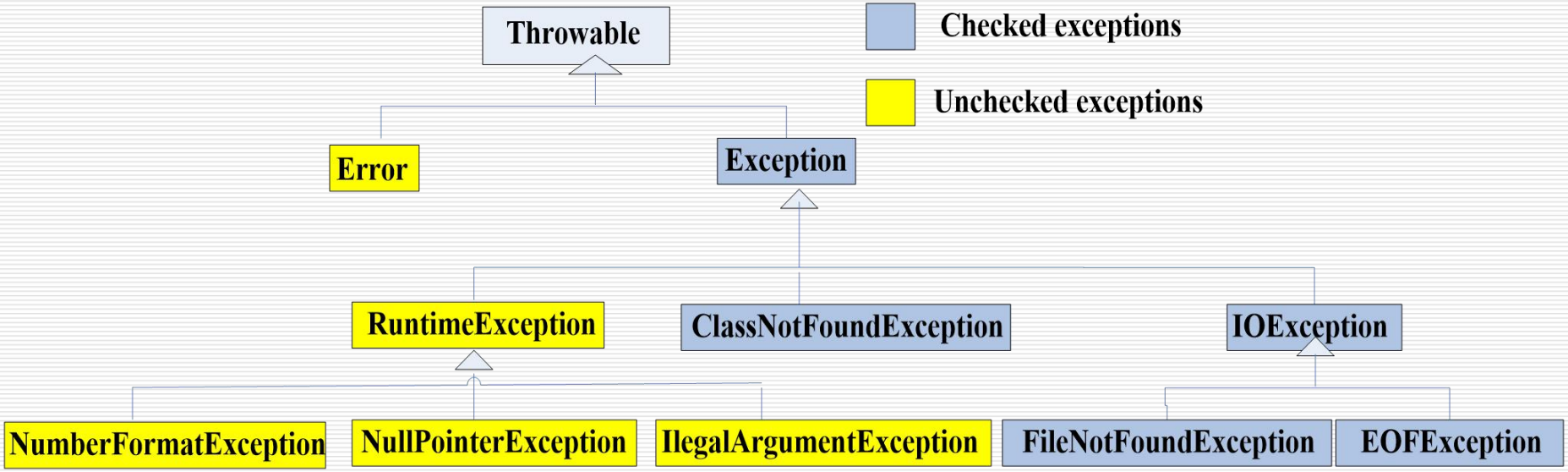
Error 动态联接失败，虚拟机错误等，java程序不应该捕获这类异常，也不会抛出这类异常

4. Exception Classes and throws (cont.)

- Java divides exceptions in two categories: *checked exceptions* and *unchecked exceptions*.

- Exceptions that are "checked" by the compiler are called *checked exceptions*.
  Java编译器要求程序必须捕获(通过try-catch)或者声明抛出(通过throws)这种异常。
- Exceptions that are not checked by the compiler are called *unchecked exceptions*.
  RuntimeException
  Java编译器允许不对它们进行处理。

4. Exception Classes and throws(cont.)



- 查看帮助文档：
  - Exception ， RuntimeException

4. Exception Classes and throws (cont.)

- If a checked exception might occur in a method,the method does not have catch block to handle the checked exception,
  - then the checked exception must be declared in the method header using a throws clause. Otherwise the compiler will not compile this file.(如果掷出多个异常，多个异常之间用”,”分割。)
  - For exampleExceptionDemo.java， *IntegerReader.java*
    - *stdIn*.readLine()
    - Integer.*parseInt()*
- 使用类的方法的时候要注意其是否抛出异常，你应如何处理？

4. Exception Classes and throws (cont.)

- 继承于RuntimeException的所有异常类皆有Java自动抛出，它所代表的是编程上的错误，在应用程序变成产品以前应该防止的设计缺陷，可能是：
  - 无法捕捉的错误(例如你的函数收到客户端传来的一个null reference)。
  - 身为程序员应该在自己的程序代码中检查的错误(例如：ArrayIndexOutOfBoundException是告诉你，你应该留意array的大小)。
- 将某些异常归类为RuntimeException，具有极大好处，因为这些异常在侦错过程中能够派上用场。
  - 一般程序通常不会捕捉Runtime Exception，程序员也可以选择掷出或捕捉某些Runtime Exception。
    - *IntegerReader.java*

4. Exception Classes and throws (cont.)

- 捕捉异常的顺序和不同catch语句的顺序有关，当捕获到一个异常时，剩下的catch语句就不再进行匹配。
  - 在安排catch语句的顺序时，首先应该捕获最特殊的异常，然后再逐渐一般化。也就是一般先安排子类，再安排父类。
  - 避免用“懒惰”的方法来捕获最通用的Exception类型，因为捕获的异常类型越具体，用于恢复的代码就越具体。

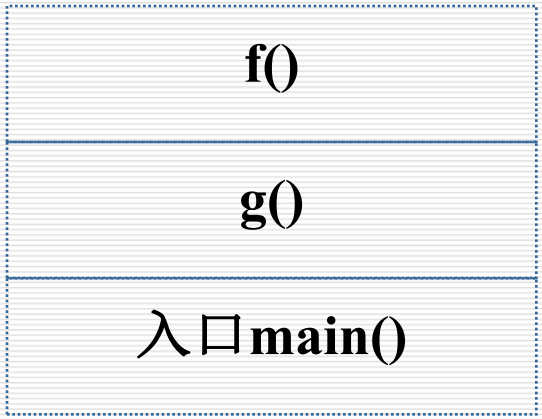
5. Exception object information

- *Exception objects* have a group of methods common to them all.
  - *String getMessage()*
    - Obtains the argument that was passed to the constructor of the exception object.
  - *String toString()*
    - The name of the exception (its type), followed by a colon (:), followed by the String returned by method getMessage.

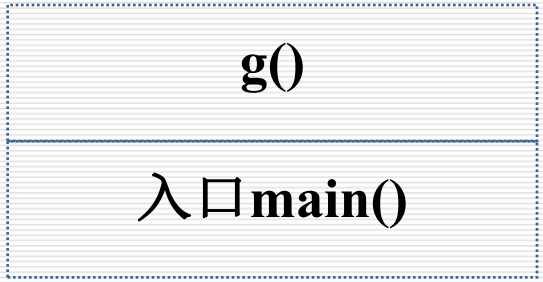
5. Exception object information(cont.)

- *void printStackTrace()*.
  - This method displays the String returned by method toString, followed by information that shows the sequence of methods that were called before the exception was thrown.
- For example ExceptionMethods.java
  - 下页图示……

5. Exception object information(cont.)



JVM通过创建调用栈来记录各个方法被调用的顺序



当方法f()执行完毕退出时，从调用栈中将其删除

6. finally

- finally 块必须与 try 或 try/catch 块配合使用
  - 无论是否发生异常，finally都一定执行（除非Java虚拟机中途终止运行）
    - 业务逻辑代码中无论发生什么都必须执行的代码，则可以放在finally块中
      - 例如：最常见的就是把关闭connection、释放资源等

7. User-Defined Exception

- A new **checked exception** class can be defined by extending the class **Exception**;
- A new **unchecked exception** class can be defined by extending the class **RuntimeException**.
- *例如：*
  - *OutOfRangeException.java*
  - EmptyException.java
  - *PositiveInteger.java*
  - *SquareRoot.java*

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

2.1.5 Javadoc

- 一个程序员编好的代码如果需要让别的程序员使用，如何提供该代码的使用说明？---说明文档
- 维护软件比开发软件花费的更多，如果软件提供了很好的说明文档，维护工作将很容易。

2.1.5 Javadoc(cont.)

- 在程序代码说明文档的撰写上，最难的问题就是文档本身的维护；
  - Java程序设计的解决办法：让程序代码和文档链接在一起。
  - **Javadoc comments are written by the programmer.**
- **Javadoc**命令会**解析 Java 源文件**中的声明和文档注释，并产生相应的 **HTML网页**（缺省），描述公有类、保护类、内部类、接口、构造函数、方法和域。

2.1.5 Javadoc(cont. syntax)

①The multi-line structure and The single line structure looks like the following:

```
/**
 * body text
 * body text
 * body text
 */
```

```
/** body text */
```

②Most Javadoc comments include Javadoc tags. Javadoc tags begin with the "at-symbol" (@), followed by the name of the tag.

2.1.5 Javadoc(cont. Common Javadoc Tags)

- @author
  - – programmer's name; used for classes
  - 格式：@author author-information
    - 可以包含电子邮箱和其它适合加入的信息，可以提供多个标签，列出所有作者。
- @version
  - – version of the program; used for classes
  - 格式:@version version-information
- 举例

2.1.5 Javadoc(cont. Common Javadoc Tags)

- @param
  - – description of a formal parameter; used for methods and constructors
  - 格式：@param parameter-name description
    - Description(描述句)是纯文本，可以延续数行，直到遇上新标签才结束；
    - 此一标签的使用次数不限，通常我们会在撰写时为每一个参数提供一份说明。
  - 举例

2.1.5 Javadoc(cont. Common Javadoc Tags)

- @return
  - – description of a return value; used for methods
  - 格式：@return description
    - 其中的description用来描述返回值的意义，可以延续数行。
  - 举例

2.1.5 Javadoc(cont. Common Javadoc Tags)

- **@exception** (or **@throws**)
  - – description of an exception; used for methods and constructors
  - 格式:
  - @throws fully-qualified-class-name description
    - description则是用来说明在什么情形下调用这个函数，会产生该类型的异常。
  - 举例

2.1.5 Javadoc(cont. Common Javadoc Tags)

- @see
  - – reference to a related entity; used for classes, constructors, methods, and data fields
  - 格式：@see classname

    .....
    - 这个标签的功能是可以通过它参考其它class的说明文档，javadoc会自动为@see标签产生一个HTML超链接，链接到其它文档。
  - 举例

2.1.5 Javadoc(cont.)

③ **Javadoc comments can contain HTML tags. Use HTML tags where appropriate.**

For example, TwoInts.java

2.1.5 Javadoc(cont.)

- **Documenting Exceptions Using Javadoc**
  - **The Javadoc tag for an exception is the @throws tag—the @exception tag can also be used.**
  - **When the method throws more than one exception, each exception should be documented on a different line and should be listed alphabetically.**

2.1.5 Javadoc(cont.)

- Place the @throws tags after the @param and @return tags.
- The @throws tags should describe the situation(s) that will cause the exception.
- For example *IntegerReaderThrowsException.java*

2.1.5 Javadoc(cont.)

- This course will only require Javadoc comments for public and protected entities—classes, constructors, methods, and variables that are preceded by the keyword public or protected.
- 详细的javadoc语法请参阅文档：
  - http://java.sun.com/j2se/javadoc/index.html

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

光明牛奶系统中部分类的实现-Product类

```
public class Product {

    /* Identification number of the product.*/
    private String code;

    /* Description of the product.*/
    private String description;

    /* Price of the product.*/
    private double price;

    /* Production date of the product.*/
    private Date productionDate;

    /* Shelf life of the product.*/
    private String shelfLife;

    /**
     * Constructs a <code>Product</code> object.
     *
     * @param initialCode  the code of the product.
     * @param initialDescription  the description of the product.
     * @param initialPrice  the price of the product.
     * @param initialProductionDate  the production date of the product.
     * @param initialShelfLife  the shelf life of the product.
     */
    public Product(String initialCode, String initialDescription, double initialPrice,
                   Date initialProductionDate, String initialShelfLife) {

        this.code = initialCode;
        this.description = initialDescription;
        this.price = initialPrice;
        this.productionDate = initialProductionDate;
        this.shelfLife = initialShelfLife;
    }
```

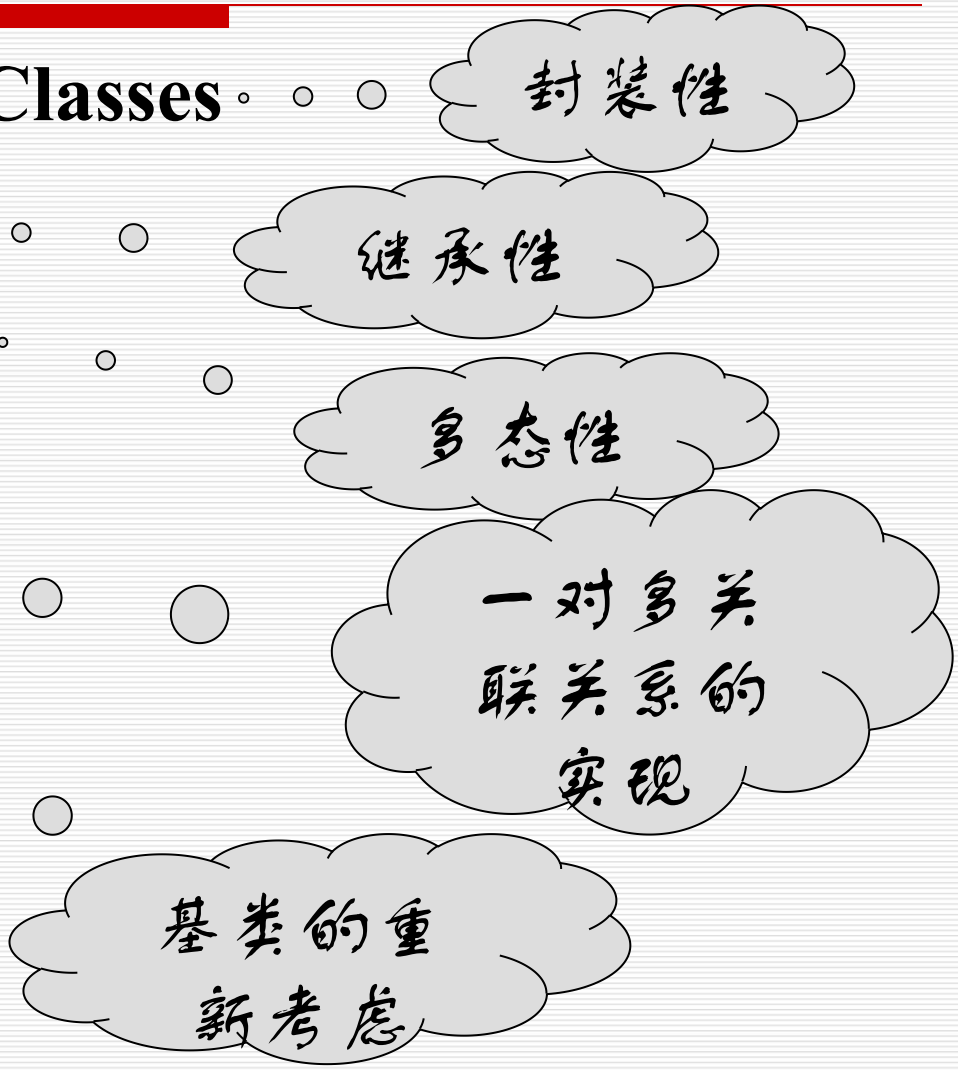
```
    /**
     * Returns the code of this product.
     *
     * @return  the code of this product.
     */
    public String getCode() {

        return this.code;
    }

    /**
     * Returns the description of this product.
     *
     * @return  the description of this product.
     */
    public String getDescription() {

        return this.description;
    }

    /**
     * Returns the price of this product.
     * @return  the price of this product.
     */
    public double getPrice() {
        return this.price;
    }

    /**
     * Returns the production date of this product.
     * @return  the production date of this product.
     */
    public Date getProductionDate() {

        return this.productionDate;
    }
```

图书馆系统中部分类的实现-Order类

```java
public class Order {

    /* Total cost of the order.*/
    private double total;

    /* List of sale items for an order.*/
    private ProductSaleList productSaleList;

    /**
     * Constructs an <code>Order</code> object.
     * <p>
     * The collection of the sale items is initially empty.
     * </p>
     * @param initialTotal  the total of the order.
     */
    public Order(int initialTotal) {
        this.total = initialTotal;
        this.productSaleList = new ProductSaleList();
    }

    /**
     * Returns the total of this order.
     * @return  the total of this order.
     */
    public double getTotal() {

        return this.productSaleList.getTotalCost();
    }

    /**
     * Returns the sale items collection.
     * @return  a {@link ProductSaleList} object.
     */
    public ProductSaleList getProductSaleList() {

        return this.productSaleList;
    }
```

Unit 2. Class Implementation

- 2.1 Implementing Classes
- ✓ 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

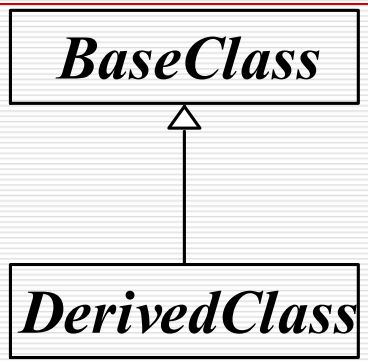


2. 2 Inheritance

- 今日问题簿：

1. 继承成员的访问权限？
2. 向上转型和向下转型的概念和特点？
3. 何为覆写？否掌握equals方法和toString方法的覆写？
4. Final关键字的用法？

1. Implementing Specialization/Generalization Relationships



- 面向对象程序的特点之二---继承性

```
public class  BaseClass {
        …
}
public class  DerivedClass  extends  BaseClass  {
        …
}
```

- 在java中，通过关键字extends表示子类(或派生类) 与父类(或超类，或基类)之间的继承关系， extends关键字后面只能跟一个基类。

1. Implementing Specialization/Generalization Relationships

- The derived class automatically inherits the properties/attributes and behaviors/methods of the base class(构造函数除外，即子类不能继承父类的构造函数).
  - Through inheritance, we can reuse and extend code that has already been thoroughly tested without modifying it.
- We may extend the base class by adding features.
  - Person-Employee.png Person.java Employee.java

1. Implementing Specialization/Generalization Relationships

- The private attribute name and address are indeed inherited-----they are part of the data structure comprising an Employee object------but they are nonetheless 'invisible' to the Employee!
- If our inclination is to make all attributes private, how can a subclass ever manipulate its privately inherited attributes?
  - The auswer is : through the public accessor/mutator methods that it has also inherited from its parent.
- For example Person.java Employee.java

1. Implementing Specialization/Generalization Relationships

- The derived class may reusing base class behaviors: using the 'super' Keyword
  1) 调用父类的构造函数，如 super([paramlist])，每个子类构造方法的第一条语句如果没有使用**super**来用父类的构造函数，那么编译器就会隐含调用**super()**，如果父类没有这种形式的构造函数，那么在编译的时候就会报错。
  2) 调用父类的方法（子类复用父类的代码），如 super.method([paramlist])，We like to avoid code duplication and encourage code reuse in an application.
     - For example Person.java Employee.java

2. 对象的生成过程

1. 类型为Employee的一个对象首次创建时，或者Employee类的静态方法或数据首次调用时，java解释器必须查找环境变量classpath所指定的位置，找出并装载Employee.class。
2. Employee.class装载后，如果它有超类，那么便会继续装载它的超类，如果超类还有超类，便会继续转载第二个超类，依次类推。从超类到Employee类的所有静态初始化模块顺序运行，静态初始化动作仅发生一次。

2. 对象的生成过程(续)

3. 当1.中是创建一个 Employee对象时，从超类到Employee类会分别顺序执行如下步骤：

   3.1 会在堆里分配一个足够的存储空间。

   3.2 系统会先用默认值初始化对象的成员变量 。

   3.3 执行所有出现于数据定义处的初始化动作。

   3.4 执行构造函数。

✓ 如果应用程序没有结束，在同一个应用程序中再创建类型Employee的对象时，则从步骤3开始顺序执行。

3. Casting objects(对象转型)

1. Upcasting（向上转型）

- **Upcasting: An subclass reference variable can be assigned to a superclass reference variable.**
  - **Upcasting is always legal**
  - **Employee employee=**

```
new Employee("Joe", "100 Ave", 3.0);
Person person = employee;
```

3. Casting objects(对象转型cont.)

- The reference person, which points to an Employee object, **cannot be used to invoke the Employee methods!**
  - For example Person.java Employee.java

```
String name = person.getName(); //legal
String address = person.getAddress();  //legal
double salary = person.getSalary(); //illegal
```

注：父类的引用变量虽然指向子类对象，但不能通过该引用调用子类所特有的方法；

3. Casting objects(对象转型cont.)

legal:

```
Employee  employee=
                    new Employee("Joe", "100 Ave", 3.0);
Person  person = employee;
double salary =  ((Employee) person).getSalary();
```

注：如果想通过该父类对象的引用调用子类所特有的方法，则必须对其进行强制类型转换

3. Casting objects(对象转型cont.)

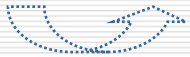
2. 向上转型(upcasting)是自然的、合法的，但是向下转型(downcasting)有着特殊性：
   - 如果父类对象引用指向的是一个子类的对象，可以使用downcasting，把超类引用显式地转型为一个子类的对象，例如：…
   - 如果超类对象引用指向的是一个自身类对象，则不可以使用downcasting，例如：…

3. Casting objects(对象转型cont.)

- **illegal:**

```
Person person =
        new Person ("Joe ", "10 Main Ave");
double salary = ((Employee) person).getSalary();
```

ClassCastException



3. Casting objects(对象转型cont.)

- **The instanceof operator is often used to avoid an illegal downcasting！（instanceof操作符与downcasting合用）**
- instanceof操作符的语法格式：

  object* instanceof *ClassX

  - 如果object是ClassX的对象引用，返回为true；
  - 如果object是ClassX的子类对象引用，返回为true；
  - 如果object是null，该表达式返回false；

3. Casting objects(对象转型cont.)

- 例如：

```
Person person =
        new Employee ("Joe Smith", "100 Main Ave“,1);
……
if (person instanceof Employee) {
   salary = ((Employee) person).getSalary();
}
```

4. overriding vs. overloading

- 两种做法可以产生derived class 与base class之间的差异：
  1. We may extend the base class by adding features.
  2. Overriding
     - 子类与父类使用相同的函数特征，但子类中方法体的实现与父类中不一样。
     - Overriding involves ‘rewriting’ how a method works internally,without changing the interface to/signature of that method.

     例如： Person.java Employee.java

4. overriding vs. overloading(cont.)

- Overriding (方法的覆写)
  - 在子类中定义一个与父类同名，返回类型、参数类型均相同的一个方法称为方法的覆写。
    - 注：不能覆写父类中声明为final或static的方法
  - For example

    Person.java

    Employee.java
- 将方法的覆写与方法的重载进行比较
  - 方法覆写(Overriding)发生在子类和父类之间
  - 方法重载发生在一个类内

5. Method equals and Method toString

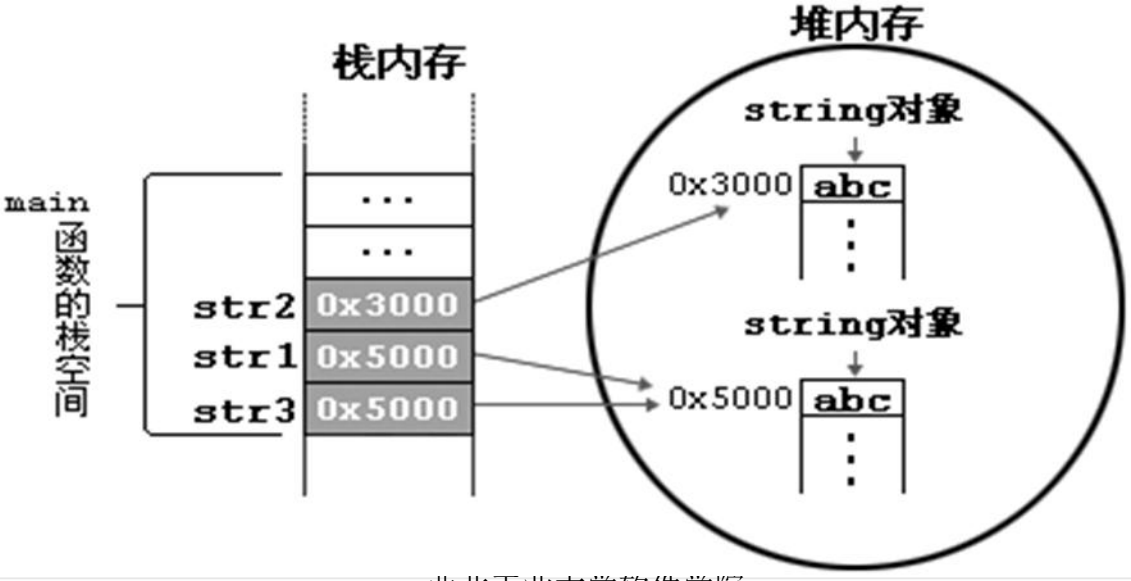
- In Java, all classes descend, directly or indirectly, from class **Object**, so all classes inherit the methods defined in class **Object**.

5. Method equals (1)

- Object 中的方法equals( )比较两个对象的引用是否相同，即比较两个对象是否是同一个对象；

  public boolean equals(Object obj){… }

  - This default implementation is exactly the same as obj1 == obj2.
    - For example Person.java Employee.java



5. Method equals(cont.)(1)

- 在大部分类中，继承自Object类的equals()方法都不适用，因此每个类都需要覆写该方法；
  - **the 'wrapper' classes(Boolean,Integer,Float,etc.); Date, String;and a few others.**
  - **CatalogItem**

```
public boolean  equals(Object  object)  {
    return object instanceof  CatalogItem &&
          getCode().equals(((CatalogItem)object).getCode());
}
```

5. Method toString(2)

- Class Object defines a method called toString that returns the string representation of the invoking object.
  - public String toString(){ }
    - The version of this method defined in class Object returns a String with the following format: *ClassName@number*

5. Method toString(cont.)(2)

- **It's a good idea to override the toString() method for all classes that you create from scratch.**
  - 在实际应用中，一般覆写的toString()方法会实现这样的功能：
    - 将一个对象有关属性信息转换成一个一定格式的实用的字符串信息；
    - CatalogItem

```
public String toString()  {
    return  getCode() + "_" + getTitle() + "_"
                + getYear() + "_" + isAvailable();
}
```

6. The 'final' Keyword

1) If we fish to assign to variable x a value that cannot ever be changed-i.e.to make x a constant-we do so by declaring the variable with the keyword final,as follows:

```
final int x = 3;
x = 4;//will produce a compilation error
```

注： public final 变量，用户可以随心所欲地使用，无法赋值，尽管是public访问权限，也没有违反Java编程的安全原则。

6. The 'final' Keyword(cont.)

- final常量
  - 在定义处初始化
  - 或在所有的构造函数中初始化
- 为了节省内存，我们通常将常量声明为静态的
  - static final常量必须在定义处初始化。

2) If we wish to prevent a subclass from overriding a method that it inherits from a bass class,we can declare the method as final in the base class:

Person.java Employee.java

6. The 'final' Keyword(cont.)

- 为了效率上的考虑，将方法声明为**final**，让编译器会自行对**final**方法进行判断，并决定是否进行优化。通常在方法的体积很小，而我们确实不希望它被覆盖时，才将它声明为**final**。

3) If we wish to prevent a class from being specialized as a subclass,we can declare an entire class to be final: Person.java

6. The 'final' Keyword(cont.)

4) If we wish for a class 'X' to be able to publish a list of valid values for use by client code as arguments to one of X's methods,we typically do so having X declare public static final attributes as constants; for example

例如：javax.swing.JFileChooser

- public void setFileSelectionMode(int mode)
  - mode - the type of files to be displayed:
    - JFileChooser.FILES_ONLY
    - JFileChooser.DIRECTORIES_ONLY
    - JFileChooser.FILES_AND_DIRECTORIES

6. The 'final' Keyword(cont.)

- Final声明可以保护public属性，与private属性有相同的安全性，因为Final的属性值无法被赋予其他新值。
- 例如：Choice.java

Unit 2. Class Implementation

- 2.1 Implementing Classes
- ✓ 2.2 Inheritance
- 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface

7. Implementing the Milk System

- 一对一关联关系的实现：关联属性转换为类的私有属性。
- 继承关系的实现：extends
- 静态类图中类的实现举例：milkSystemSolution.png
  - Product
  - Jelly
  - PureMilk
  - Yogurt
  - MilkDrink

7. Implementing the Milk System-Recording

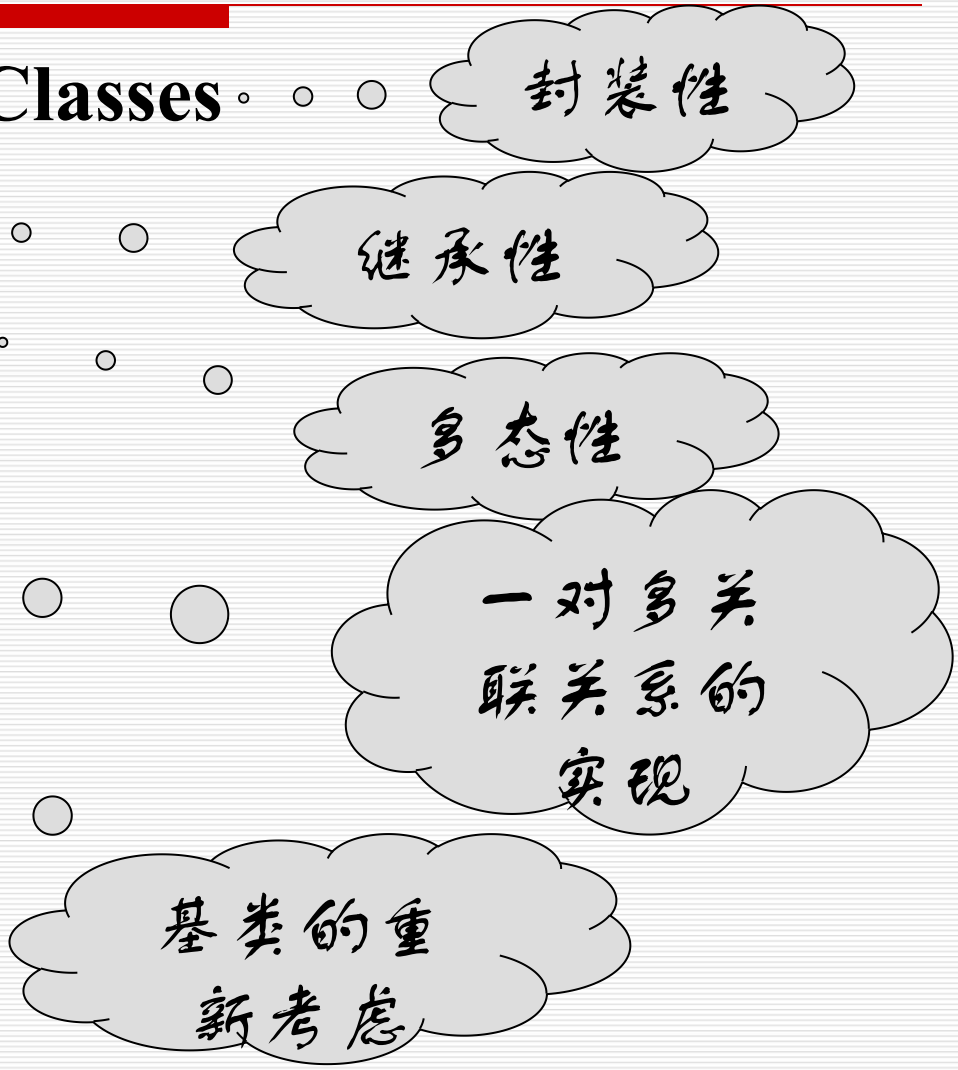
```
public class Jelly extends Product {

    /* Content of flavor in jelly.*/
    private String flavor;

    /**
     * Constructs a <code>Jelly</code> object.
     *
     * @param initialCode  the code of the jelly.
     * @param initialDescription  the description of the jelly.
     * @param initialPrice  the price of the jelly.
     * @param initialProductionDate  the production date of the jelly.
     * @param initialShelfLife  the shelf life of the jelly.
     * @param initialFlavor  the content of flavor in jelly.
     */
    public Jelly(String initialCode, String initialDescription, double initialPrice,
                 Date initialProductionDate, String initialShelfLife, String initialFlavor) {

        super(initialCode, initialDescription, initialPrice, initialProductionDate, initialShe

        this.flavor = initialFlavor;
    }
```

7. Implementing the Library System-Recording

```java
/**
 * Returns the content of flavor in jelly.
 *
 * @return  the content of flavor in jelly.
 */
public String getFlavor() {

    return this.flavor;
}

/**
 * Returns the string representation of this jelly.
 *
 * @return  the string representation of this jelly.
 */
@Override
public String toString() {

    return super.toString() + "_" + getFlavor();
}
```

Unit 2. Class Implementation

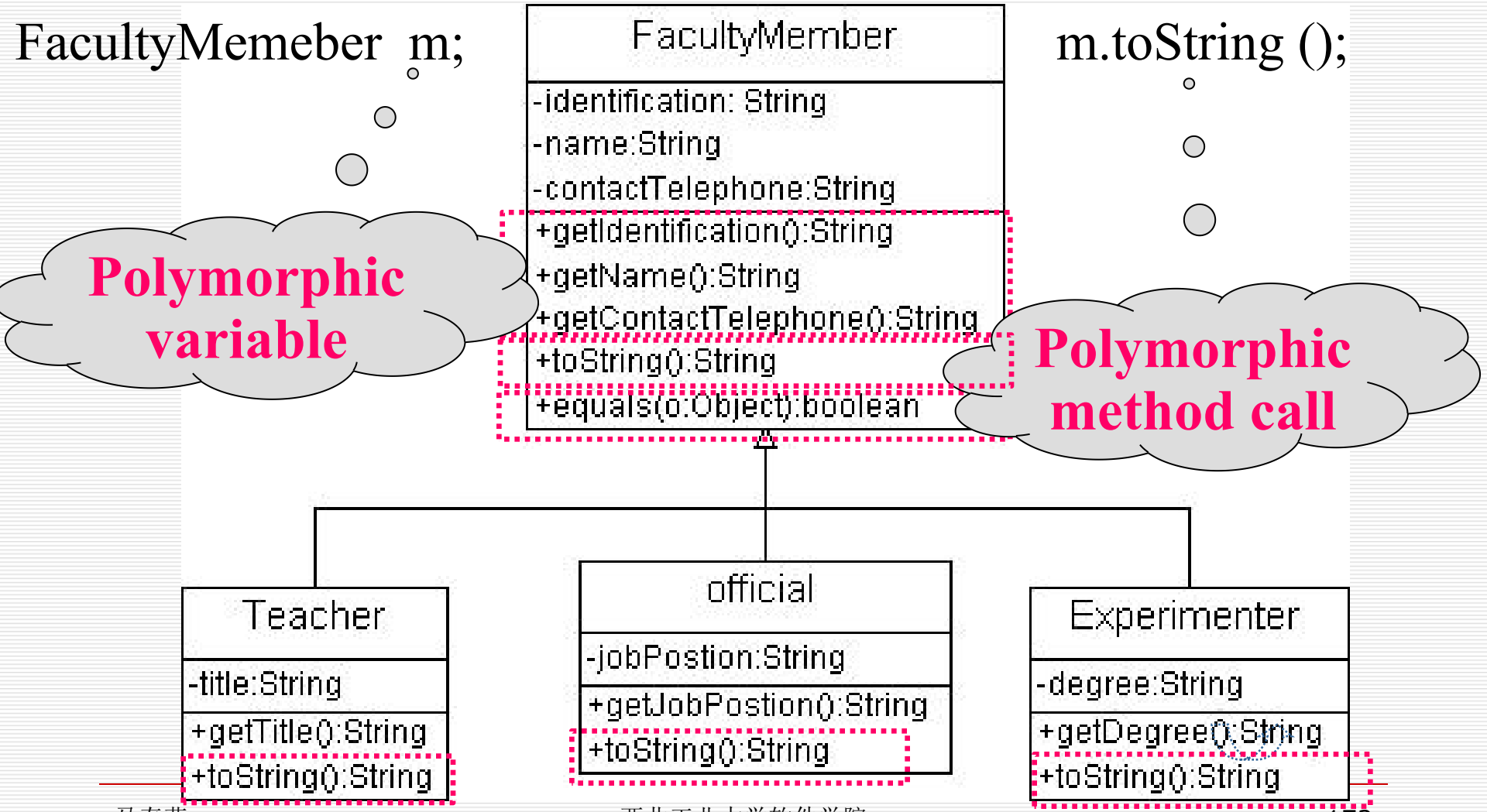
- 2.1 Implementing Classes
- 2.2 Inheritance
- ✓ 2.3 Polymorphism
- 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface



2.3 Polymorphism

- The term polymorphism is defined in Merriam-Webster's dictionary as "the quality or state of being able to assume different forms."
- In an Object-Oriented Programming Language, polymorphism is One Distinguishing Feature.
  1. Polymorphic variable
  2. Polymorphic method call
  3. Discussion

2. Polymorphic method call



2. Polymorphic method call (cont)

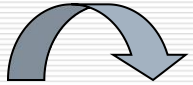
- The line of code m.toString( ) said to be ***polymorphic*** because the logic performed in response to the same method call can take many different forms, depending on the class identity of the object at run time.

2. Polymorphic method call (cont)

- How Polymorphism Works Behind the Scenes (Static vs. Dynamic Binding)
  - Static Binding----**compile-time**
    - `FacultyMember m;`
  - Dynamic Binding---**run-time**
    - Java Virtual Machine (JVM) determines which version of the method to execute by examining the reference variable (polymorphic variable) used in the method call.

3. Discussion

- **Polymorphic variable allows**
  - many types (derived from the same base type) to be treated as if they were one type, and a single piece of code to work on all those different types equally.
- **polymorphic method call allows**
  - one type to express its distinction from another, similar type, as long as they're both derived from the same base type.
- **For example FacultyDemo.java**



3. Discussion (cont.)

- **Class diagram designing**
  - **The design solution of Software College Management System (SCMS)**
    - One class Associates with all subclasses of some base class--------> One class Associates with the base class

3. Discussion (cont.)

- **Polymorphism Simplifies Code Maintenance**
  - Polymorphism can make our code much more concise.
    - If the programming language ***doesn't*** support polymorphism
      - We have to do with a variety of different kinds of faculty members using a series of if tests.
      - For example

3. Discussion (cont.)

- **Polymorphism Simplifies Code Maintenance**
  - Better still, polymorphic client code is robust to change.
    - For example, long after our **SCMS** application has been coded, tested, and deployed, we derive classes called **Administrator** and **DeanOfficial** from **official**, each of which in turn overrides the printInfo method of **official** to provide its own "flavor" of print functionality. We're now free to randomly insert **Administrators** and **DeanOfficials** into our **Faculty** collection, our polymorphic code for iterating through the collection ***doesn't have to change***!

图书馆系统中的多态

```
77                                  "Craig Larman", 656));
78          catalog.addItem(new Book("B014", "The Little Schemer", 1995,
79                                  "Daniel P. Friedman", 216));
80          catalog.addItem(new Book("B015", "Agile Software Development", 2001,
81                                  "Alistair Cockburn", 256));
82
83          catalog.addItem(new Recording("R001", "Getz/Gilberto", 1963,
84                                  "Stan Getz and Joao Gilberto", "CD"));
8                                   ording("R002", "Kind of Blue", 1997,
                                    "Miles Davis", "CD"));
                                    rding("R003", "Supernatural", 1999,
                                    "Santana", "Tape"));
```

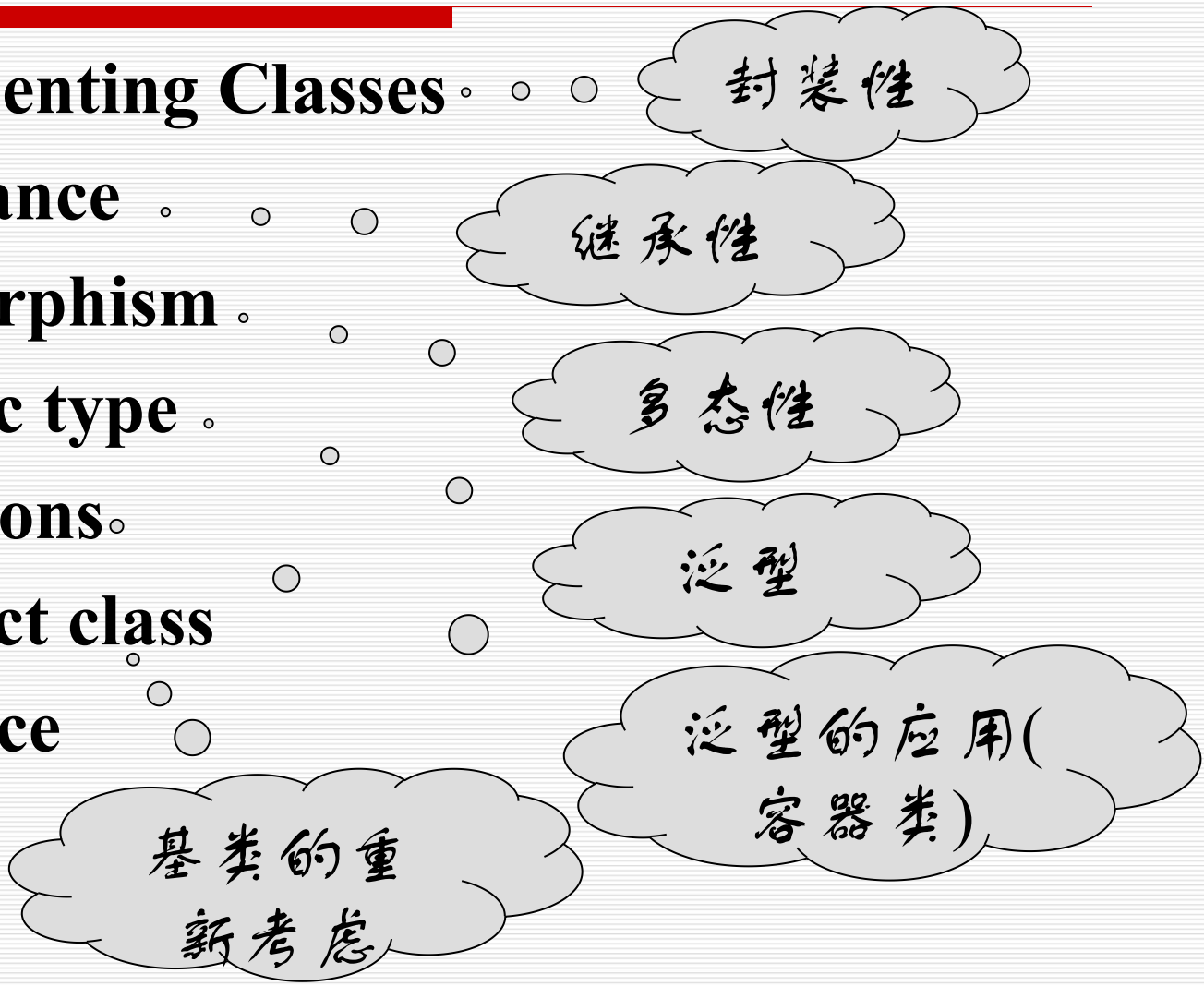
Polymorphic variable

```
228     private void displayCatalog() {
229
230
231         if (this.catalog.getNumberOfItems() == 0) {
232             stdErr.println("The catalog is empty");
233         } else {
234
235             for (CatalogItem item : this.catalog) {
236                 stdOut.println(item.toString());
237             }
```

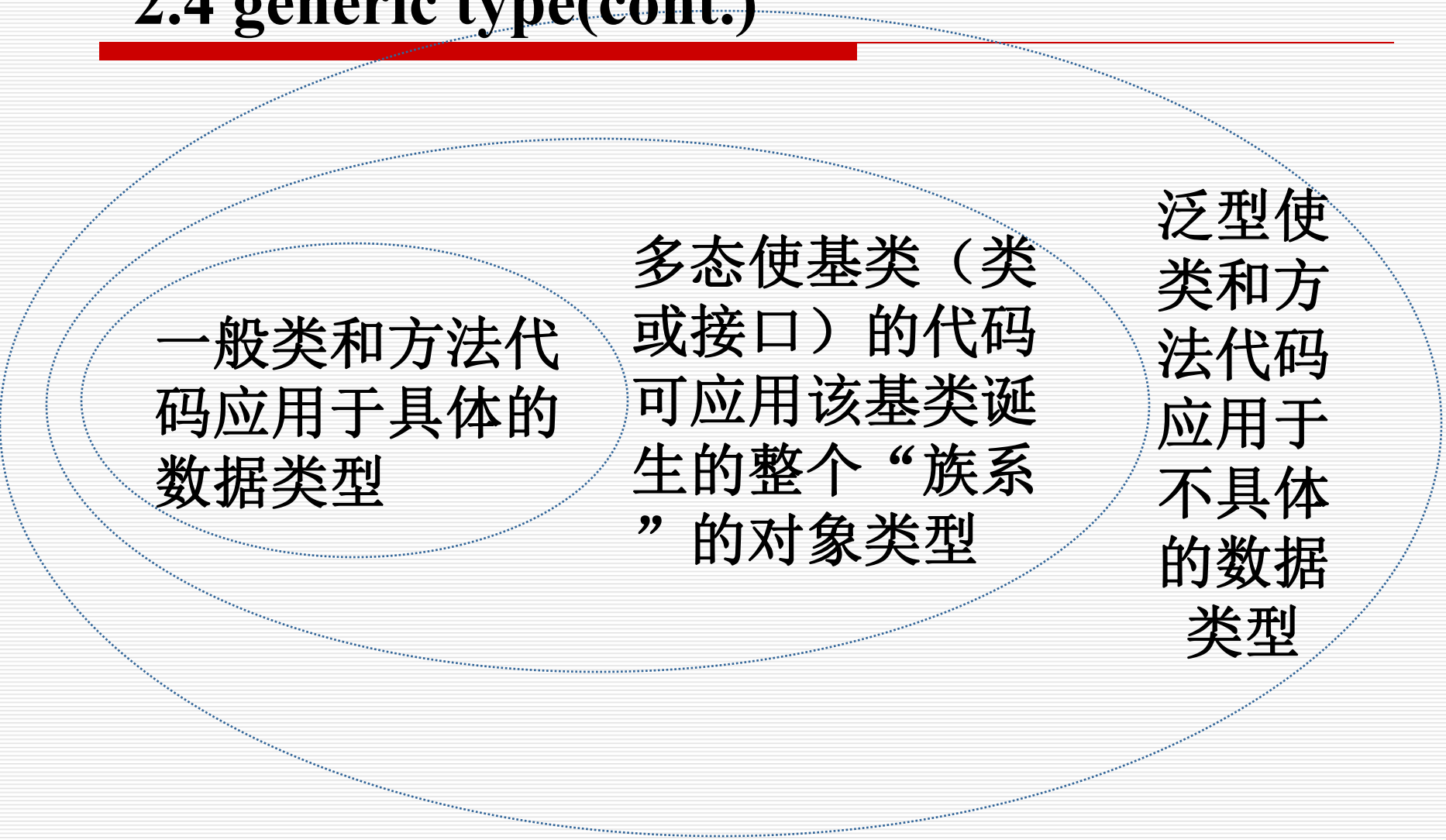
Polymorphic method call

Unit 2. Class Implementation

- 2.1 Implementing Classes
- 2.2 Inheritance
- 2.3 Polymorphism
- ✓ 2.4 Generic type
- 2.5 Collections
- 2.6 Abstract class
- 2.7 Interface



2.4 generic type(cont.)



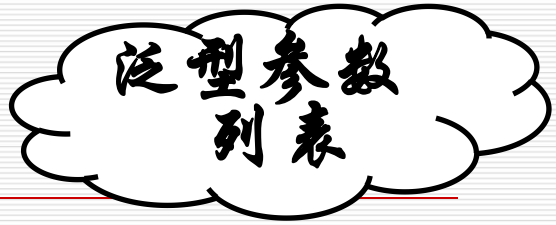
2.4.1 What is “Generic type”

- 泛型，也即“参数化类型”，就是将类型由原来的具体的类型参数化，是Java可复用性的又一大体现。
- Java 泛型有泛型接口，泛型类和泛型方法。
- Collections(Java容器类)是Java泛型类中的典型代表，常用的ArrayList<E>, Vector<E>, Map<A, B>等都是出自Collections。

2.4.2 Generic Classes (/abstract class/interface)

- 如何构建一个泛型类？
  - (1) 先写出一个实际的类。
  - (2) 将此类中准备改变的类型名(如String要改变为Point2D或Product)用一个泛型参数(如上述E，A，B等)代替，使该类成为一个泛型类。
  - (3)泛型类中的方法可以使用泛型参数，static方法无法访问该泛型类的泛型参数
  - (4)使用泛型类时，必须在创建对象的时候指定泛型参数的值
- TestPair.java

2.4.3 Generic Methods



- Methods can be parameterized even if the class they are in isn't. In these cases, the method descriptors just have to contain the type parameter in angle brackets much like the parameterized classes do(The Generic Parameter list is comma separated.).
  - public <K> K someMethod() { ... }
    - Example: GenericMethods.java ； Comparison.java
    - 调用泛型方法与调用普通方法一样，不必指明参数类型，泛型方法可以看做被无限次重载过；

2.4.3 Generic Methods（cont.）

- 如果使用泛型方法可以取代将整个类泛型化，就应该仅使用泛型方法；
- 对于static方法，无法访问泛型类的类型参数，如果需要泛型能力，必须使其成为泛型方法；

Unit 2. Class Implementation

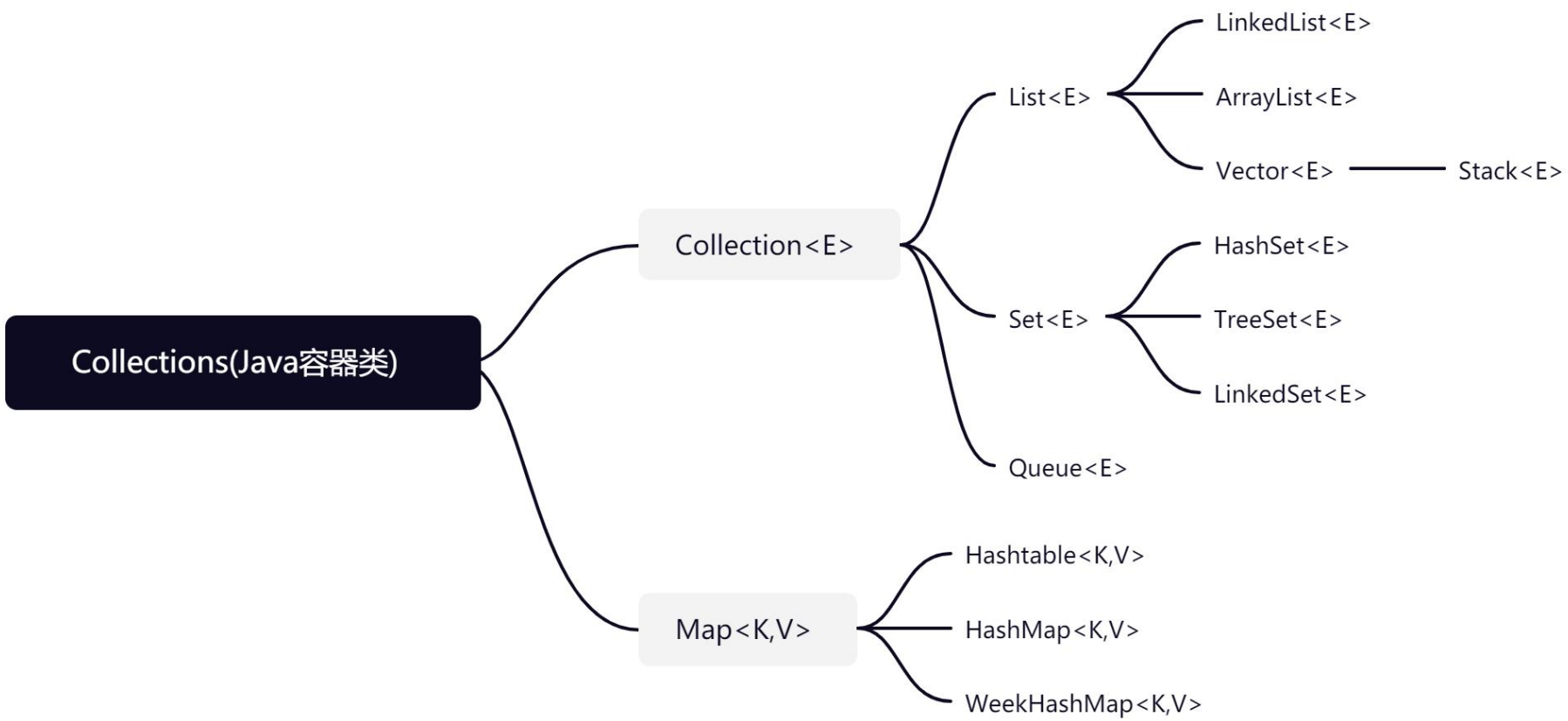
- ➢ 2.1 Implementing Classes
- ☐ 2.2 Inheritance
- ☐ 2.3 Polymorphism
- ☐ 2.4 Generic type
- ✓ 2.5 Collections
- ☐ 2.6 Abstract class
- ☐ 2.7 Interface

- For example employee-salary1.png
- We'd like a way to gather up objects as they are created so that
  - **We can manage them as a group and operator on them collectively,**
  - **We can refer to them individually when necessary.**

2.5 Collections—最具重用性的类库之一

- 写程序时有时并不确定需要管理多少对象，Java提供了一套容器类库(java.util包)用以解决这个问题，包括Collection<E>和Map<K,V>两类容器:
- 属于Collection<E>类型者，其内的每个位置仅持有一个元素，提供了add()方法添加元素，这一类型包括：
  - List<E>： 以特定次序存储一组元素；例如ArrayList<E>，Vector<E>都是一种List<E>。
  - Set<E>：元素不得重复。
- Map<K,V>所持有的则是一群成对的key-value(键值-实值对)，像个小型数据库，提供了put( )方法添加元素。
  - container.png PrintingContainers.java

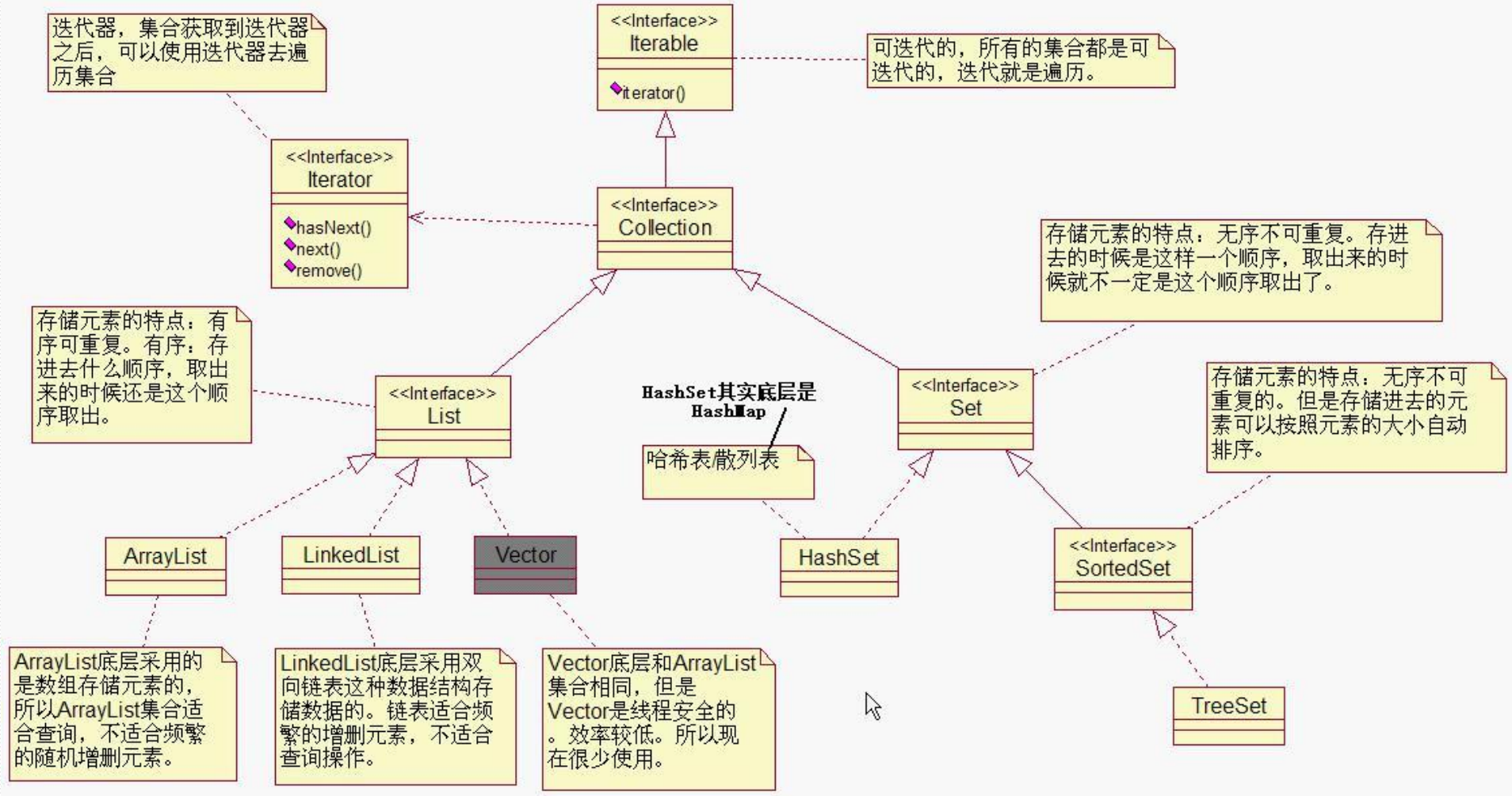
Java 简化版集合类框架图



2.5.1 Collection

- Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素（Elements）。
- Java SDK不提供直接继承自Collection的类，Java SDK提供的类都是继承自Collection的“子接口”如List和Set。
- 实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。
- Set容器类主要有HashSet和TreeSet等。

Collection类框架图



2.5.1 Collection——LinkedList类

- LinkedList类是基于链表的数据结构实现的，提供了基于头插(addFirst)和尾插(addLast)的增加，删除元素操作。
- 因其基于链表，所以写动态插入和删除密集型程序中使用LinkedList比ArrayList效率要高些。
- LinkedList和ArrayList都是非同步的方法，线程不安全，若多个线程同时访问一个List，则需要自行实现访问同步。
- LinkedListTest.java (例子：寻找最小值)

2.5.1 Collection——ArrayList类

- ArrayList是基于数组实现的，每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小，这个容量可随着不断添加新元素而自动增加，当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。
- 因ArrayList基于数组，所以适合随机查找和遍历中使用ArrayList比LinkedList效率要高些。

Array和ArrayList的区别与联系

- 区别：
  - array可以保存基本类型和对象类型，arrayList只能保存对象类型
  - array数组的大小是固定的不能更改，而ArrayList的大小可以改变
  - ArrayList有更加丰富的方法如addAllA()、removeAll()、iterator()等
- 联系：Array是数组，ArrayList是Array的加强版

运用ArrayList实现一对多的关联关系

- 现有一个需求说明：一个学生管理系统可以管理很多学生，每个学生有姓名，学号，性别，年龄信息。需要完成以下系统功能。
  - 可以增加一个学生信息
  - 可以根据学号查询某个学生的信息
- StudentManage.java(例子：实现一对多关联关系)

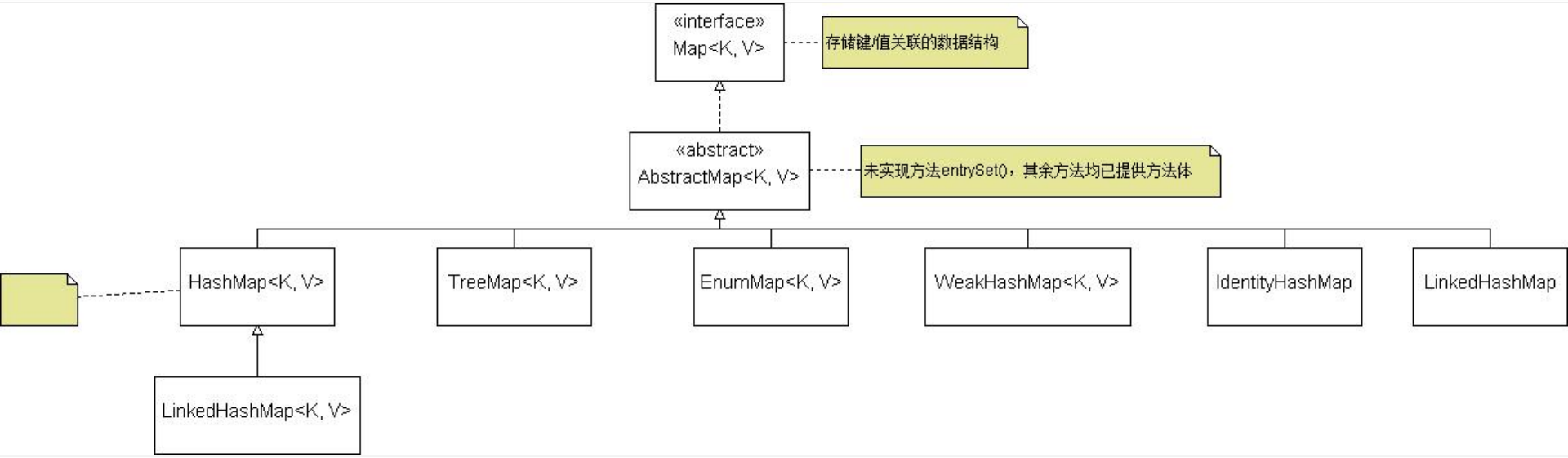
ArrayList、Vector和LinkedList

- ArrayList是最常用的List实现类，内部通过数组实现的，它允许对元素进行随机访问。数组的缺点是每个元素之间不能有间隔，当数组大小不满足时需要增加存储能力。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。
- Vector与ArrayList一样，也是通过数组实现的，不同的是它支持线程的同步，即某一时刻只有一个线程能够写Vector，但实现同步需要很高的花费，因此，访问它比访问ArrayList慢。
- LinkedList是采用链表结构存储数据的，很适合数据的动态插入和删除，但随机访问和遍历速度比较慢。另外，他还提供了List接口中没有定义的方法，专门用于操作表头和表尾元素，可以当作堆栈、队列和双向队列使用

2.5.2 Map

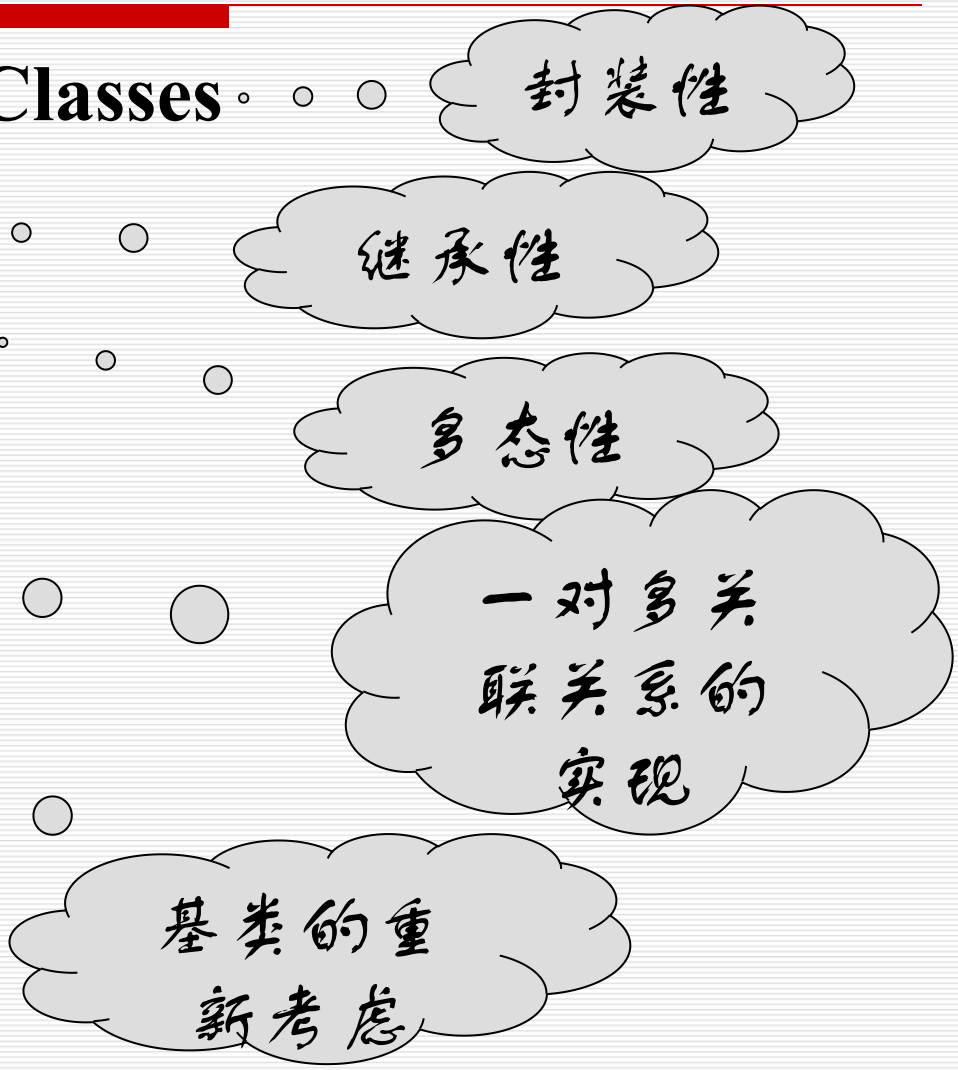
- Map没有继承Collection接口，Map提供key到value的映射。
- 一个Map中不能包含相同的key，每个key只能映射一个 value。Map接口提供3种集合的视图，Map的内容可以被当作一组key集合，一组value集合，或者一组key-value映射。
- Map中常用的两个实现是HashTable和HashMap。
- HashMapTest.java (例子：统计频率)

Map类框架图


«interface»
Map<K, V>
存储键/值关联的数据结构
«abstract»
AbstractMap<K, V>
未实现方法entrySet()，其余方法均已提供方法体
HashMap<K, V>
TreeMap<K, V>
EnumMap<K, V>
WeakHashMap<K, V>
IdentityHashMap
LinkedHashMap
LinkedHashMap<K, V>

Unit 2. Class Implementation

- ➢ 2.1 Implementing Classes
- ☐ 2.2 Inheritance
- ☐ 2.3 Polymorphism
- ☐ 2.4 Generic type
- ✓ 2.5 Collections
- ☐ 2.6 Abstract class
- ☐ 2.7 Interface



2.5.2 Arrays

今日问题簿：
数组如何声明、如何使用？

- In Java,
  - The length of an array object can never change.
  - The indexes in an array of *n* elements will range from 0 to *n* - 1.
    - The Java Virtual Machine (JVM) will throw an ArrayIndexOutOfBoundsException if a program tries to use an invalid index.
  - An array can contain a set of primitives or a set of object references.
  - 当管理一大群对象(尤其是基本数据类型的数据)，第一选择应该是数组，除非不知道究竟需要多少对象。

1. Declaring

- 声明一个存储基本类型数据或对象的数组：
  - int[] ages;
    - ages: int[ ]类型的对象(变量)
    - 数组元素的类型是int类型
  - String[] names;
    - names: String[ ]类型的对象 (变量)
    - 数组元素的类型是 String类型的
- ages and names are reference variables and their value are initially null.

1. Declaring(cont.)

- With an Array,you declare it to hold items(objects references or simple data types) of a certain type-e.g.

1. Declaring(cont.)

- Java arrays are objects,they must be instantiated using the new operator;
  - int[ ] x = new int[20]
- 当数组对象被创建的时候，Java保证该数组的内容一定会被初始化。
  - int[] ages = new int[5];
  - String[] names = new String[3];

Five zeroes

Three null values

不允许声明时指定数组大小

1. Declaring(cont.)

- The array declaration can include an initializer, a comma-separated list of initial element values within braces.
  - int[] ages = {21, 19, 35, 27, 55};
  - String[] names = {"Bob", "Achebe", null};
  - String[] names = new String[ ] {"Bob", "Achebe", null }
  - Point2D[] points = new Point2D[]{new Point2D(1,1),new Point2D(2,3)}
- 对数组元素的操作
  - 排序：Arrays.sort(…),
  - 复制：System.arrayCopy(…)

2. Using Arrays

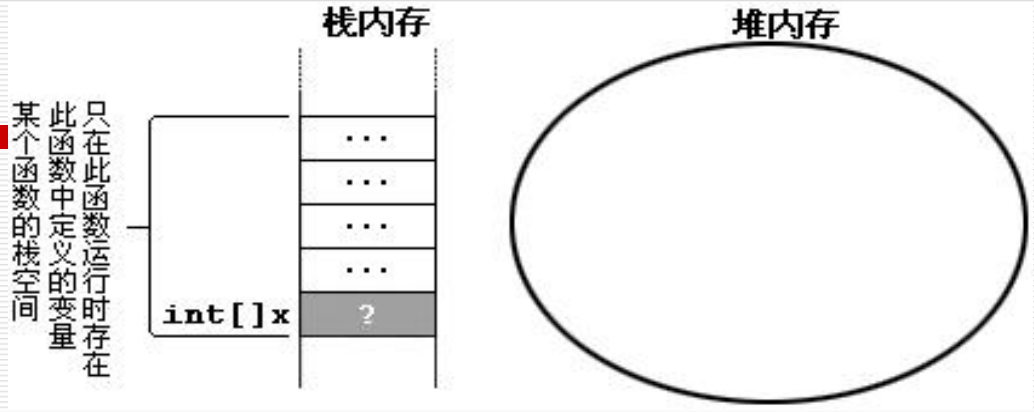
- An array object contains a public instance variable called length, which indicates the number of elements in the array.

```
int[] ages = new  int[5];
for(int index = 0; index < ages.length; index++){
        int x = ages[index];
        …
}
for(int x:ages){
        …
}
```

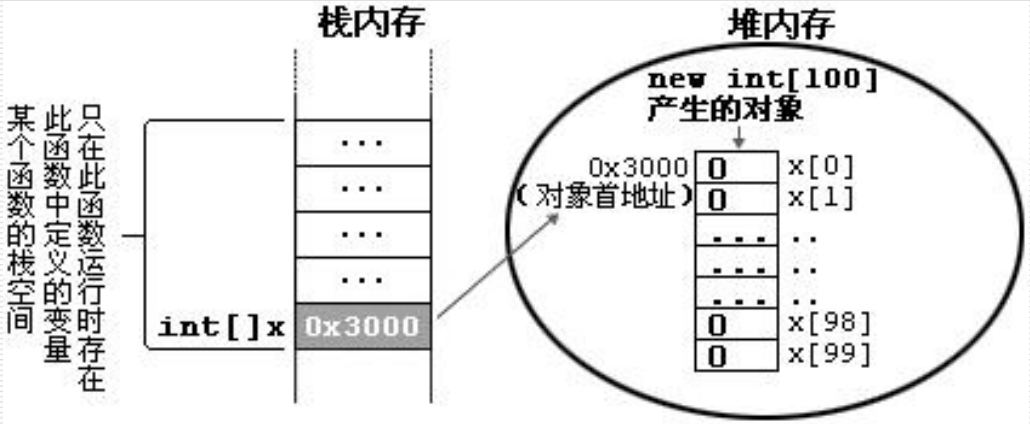
- ArraySizeBasic.java ArraySizeObject.java

无论所使用的数组类型为何，数组名称本身实际上是个reference，指向堆内的某个对象。

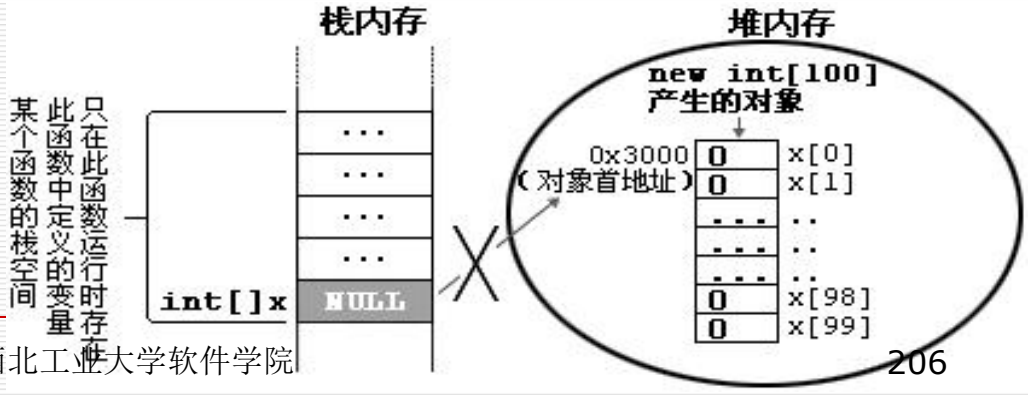
int[] x;



x = new int[100];

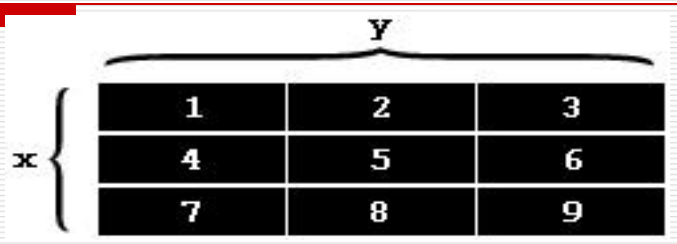


x=null;

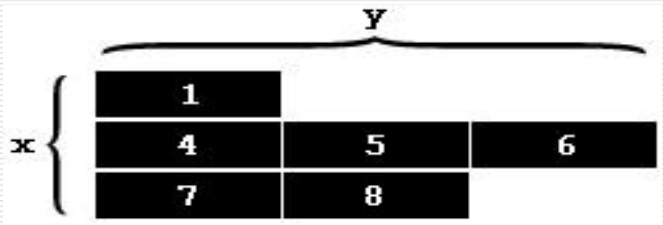


多维数组

- 如何理解多维数组，类似我们的棋盘。

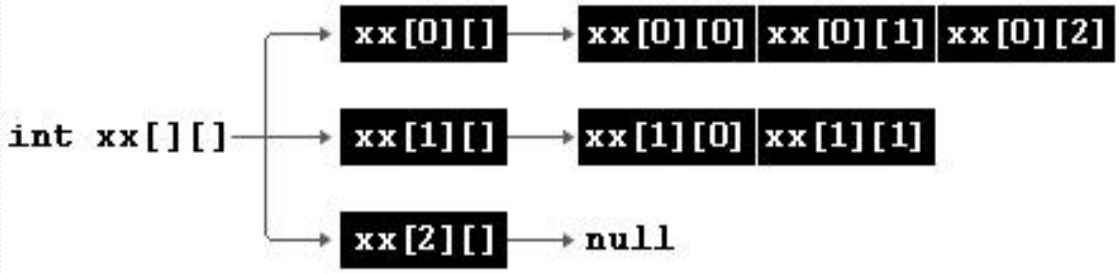


- java 中并没有真正的多维数组，只有数组的数组， Java中多维数组不一定是规则矩阵形式。



- 定义一个多维数组

```
int[][] xx;
xx=new int[3][];
xx[0]=new int[3];
xx[1]=new int[2];
```



Unit 2. Class Implementation

- ➢ 2.1 Implementing Classes
- ☐ 2.2 Inheritance
- ☐ 2.3 Polymorphism
- ☐ 2.4 Generic type
- ✓ 2.5 Collections
- ☐ 2.6 Abstract class
- ☐ 2.7 Interface

2.5.1 ArrayList and Iterators

1. The ArrayList Class
2. Iterator
3. Using the For-Each Loop in Collections

今日问题簿：

遍历ArrayList容器有哪三种方法？

1. The ArrayList Class

- java.util.ArrayList<E>
  - ArrayList<E> a = new ArrayList<E>();
    - E在此处代表指定a中只能存放E类型的对象，否则，编译器将会标记这是一个错误。
    - 存储的对象的个数没有限制；

- ***The following is a list of some of the methods defined in class JDK1.5 ArrayList<E>:ArrayList.doc***

  - ***ArrayList()*** **Constructs an empty vector.**
  - ***int size()*** **Returns the number of elements in this list.**
  - ***boolean add(E o)*** **Appends the specified element to the end of this list.**
  - ***E get(int index)*** **Returns the element at the specified position in this list.**
  - ***boolean remove(Object o)*** **Removes a single instance of the specified element from this list, if it is present (optional operation).**

- 例如：**ArrayListExample.java**

1. The ArrayList Class（cont.）

- 遍历ArrayList容器所容纳的对象序列
  - 对于ArrayList<E>，可以通过索引对其进行插入对象和取出对象的操作。

```
ArrayList<Person> people = new ArrayList<Person> ( );
                    …
for (in i=0;  i<people.size( );  i++) {
                    Person person = people.get(i);
                                …
}
```

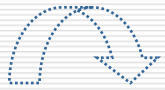
  - 通过size()和get()方法可以实现对容器ArrayList<E>的遍历。

Unit 2. Class Implementation

- ➢ 2.1 Implementing Classes
- ☐ 2.2 Inheritance
- ☐ 2.3 Polymorphism
- ☐ 2.4 Generic type
- ✓ 2.5 Collections
- ☐ 2.6 Abstract class
- ☐ 2.7 Interface

2. Iterator

- An iterator is an object whose job is to move through a sequence of objects and select each object in that sequence without the client programmer knowing or caring about the underlying structure of that sequence.
- 所有继承Collection<E>的容器（例如：ArrayList<E> ）都提供了一个方法可以返回一个Iterator<E> 对象，通过该迭代器对象可以实现遍历Collection<E>容器：
  - public Iterator<E> iterator()



2. Iterator (cont.)

- java.util.Iterator<E> provides the following methods:
  - *Boolean hasNext()*. Returns true if the iteration has more elements.
  - *E next()*. Returns the next element in the iteration.
  - *void remove()*. Removes from the vector the last element returned by the iterator.
    - 在使用调用remove之前必须先调用一次next方法，因为next就像是在移动一个指针，remove删掉的就是指针刚刚跳过去的元素。即使是你想连续删掉两个相邻的元素，也必须在每次删除之前调用next。
- For example ArrayListExample.java

使用iterator的注意事项：

```
import java.util.*;
public class test {
    public static void main(String[] args){
        ArrayList<String>  list= new ArrayList<String>();
        list.add("Vectors"); list.add(" and "); list.add("Iterators");
            String result = "";
        Iterator<String>  iterator = list.iterator() ;
        while (iterator.hasNext()) {
            result =  iterator.next();
            iterator.remove();//允许
            list.add("cat"); //不允许
        }
        System.out.println(result);
  }
}
```

3. Using the For-Each Loop in Collections

The for-each loop provides a simple way to iterate through the elements of a collection.语法如下：

```
Collection<Point2D>    c;
for (Point2D    point :c) {
        int x = point.getX();//对于容器 c中存储的每一个元素
                                point，作如下处理……
}
```

- ✓ **c:容器类型的变量**
- ✓ **point：Point2D类型的变量，每次循环的执行都从容器中取出对象并赋给该变量**

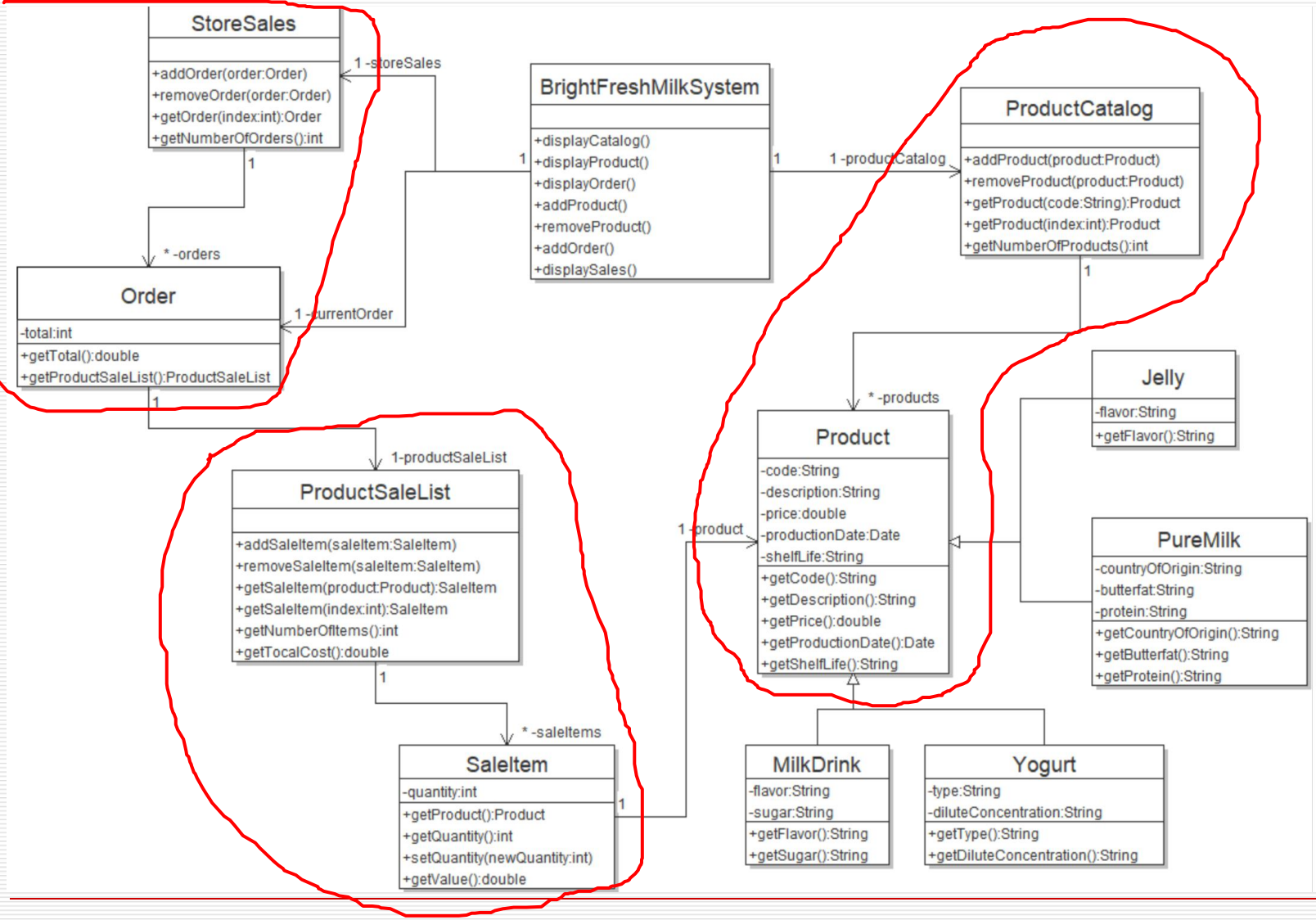
3. Using the For-Each Loop in Collections(cont.)

```
ArrayList<String> list = new ArrayList<String>();
list.add("ArrayList");
list.add(" and ");
list.add("for-each");
String result = "";
for (String element : list) {
      result += element;
}
stdout.println(result);
```

Using the For-Each Loop in Collections

- 使用For-Each循环的四种情况：
  - Collection容器类： ArrayList
    - arrayListExample.java;
  - 对数组的遍历也可以使用for-each循环
  - The for-each loop can be used only on those classes that implement the interface java.lang.Iterable<T>.
    - 举例：Client.java BankAccount.java TestClient.java
  - 可变参数

图书馆系统1对多关联关系的实现



图书馆系统1对多关联关系的实现

```java
public class ProductCatalog implements Iterable<Product>{

    /* Collection of <code>Product</code> objects.*/
    private ArrayList<Product> products;

    /**
     * Constructs an empty product catalog.
     */
    public ProductCatalog() {

        this.products = new ArrayList<Product>();
    }

    /**
     * Adds a {@link Product} object to this product catalog.
     *
     * @param product  the {@link Product} object.
     */
    public void addProduct(Product product) {

        this.products.add(product);
    }

    /**
     * Removes a {@link Product} object from this collection.
     *
     * @param product  the {@link Product} object.
     */
    public void removeProduct(Product product) {

        this.products.remove(product);
    }

    /**
     * Returns an iterator over the products in this product catal
     *
     * return  an {@link Iterator} of {@link Product}
     */
}
```

```java
public Iterator<Product> iterator() {

    return this.products.iterator();
}

/**
 * Returns the {@link Product} object with the specified
 * <code>code</code>.
 *
 * @param code  the code of an item.
 * @return  The {@link Product} object with the specified
 *          code. Returns <code>null</code> if the object with
 *          the code is not found.
 */
public Product getProduct(String code) {

    for (Product product : this.products) {
        if (product.getCode().equals(code)) {

            return product;
        }
    }

    return null;
}

/**
 * Returns the number of products in the product catalog.
 *
 * @return the number of {@link Product} objects in this product catalog.
 */
public int getNumberOfProducts(){
    return this.products.size();
}
```