Java面向对象的程序设计笔记

- 教师：吴桂兴
- 成绩：考勤+作业50%，考试50%
  - leetcode 10题 写报告
- 平时作业
  - 第一页名字
  - 第二页目录
  - 小四，行距20磅
  - 每题重新起页

常用单词

1. ambiguity　歧义
2. braces　大括号{ }
3. constant　常数
4. converter　转换器
5. deprecation　弃用
6. explicitly　显示地
7. implicitly　隐式地
8. increment　递增
9. instant　实例
10. invoke　调用（函数）
11. local variables　局部变量
12. method invocations　方法调用
13. modifier　修饰符
14. nested classes　内部类，嵌套类
15. parentheses 小括号（）
16. parameter 形参
    - argument 实参
17. reassemble　重组

18. retention　保留
19. retrieve　检索
20. subclass　子类
21. superclass　父类，基类
22. syntax　语法

- 栈
  - 局部变量
- 堆
  - 成员变量
- 方法区（数据段）
  - 静态变量
- 值传递和引用传递
- 先实例化外部类，再实例化内部类
- this指当前类，必须先有类，不能在method中直接用this
- 只能导入整个包或者单一包，不能导入包的某个子集
- UML 继承 接口 依赖的符号
  - https://blog.csdn.net/bendanbaichi1989/article/details/7661874

---

1 课程介绍

- 课程重点
- An introduction to Java
  - – Java history, about the Java technology, object-oriented programming concepts
- Learing the Java lanuage
  - – Language basics, classes and objects, annotations, interface and inheritance, number and strings, generics, package
- Essential Java Classes
  - – Exceptions, Basic I/O, concurrency
- Collections
- Custom networking

---

2 面向对象编程概念

- Object = 状态state + 行为behavior
  - 域 field(variables)
  - 方法 methods
- 好处
  - 模块化
    - 源代码独立编写和维护
    - 一旦创建，可轻松传递对象
  - 信息隐藏
    - 内部实现细节隐藏
  - 代码重用，可实现/测试/调试复杂对象
  - 调试简便，可删除和替换某特定对象
- 接口
  - 接口实现方式在编译时强制执行
  - API　Application Programming Interface

---

3 java语言基础

变量类型

- 实例变量（非静态字段）
  - 它们的值对于类的**每个实例都是唯一的**
- 类变量（静态字段）
  - 无论实例化多少次，该变量**只有一个副本**
  - static int numGears = 6；创建静态字段，final代表数值永远不改变
- 局部变量
  - 声明字段，仅对声明它们的方法可见
  - 包括：形参，方法局部变量，代码局部变量
  - 局部变量将**临时状态存储在方法内部**
- 参数
  - 参数的分类始终是变量variables，而不是字段fields

变量名称

-

以字母, $, _下划线开头，惯例以字母开头

- 不允许使用空格
- 选变量名，用完整单词而不是缩写
- 变量名不是关键字 keyword 或保留字 reserved word
- 命名法
  - 变量 gearRatio
  - 常量值 NUM_GEARS 按惯例，下划线只在该情况使用

Java Language Keywords

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

基础数据类型

- byte 1B -128~127
- short 2B -32768~32767
- int 4B Integer类将int作为无符号整数
- long 8B
- float 4B 不可用于精确值，如货币，需用java.math.BigDecimal Numbers和Strings类有BigDecimal
- double 8B
- boolean 只有true和false
- char 2B 16位Unicode字符， 最小值为'\ u0000'（或0），最大值为'\ uffff'
- String 不是基础数据类型，字符串对象一旦创建值无法更改
- 已声明未初始化的字段field，由编译器默认初始化，**局部变量不会默认初始化**

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

- \b (backspace),
- \t (tab),
- \n (line feed),
- \f (form feed),
- \r (carriage return),
- \" (double quote),
- \' (single quote),
- and \\ (backslash).

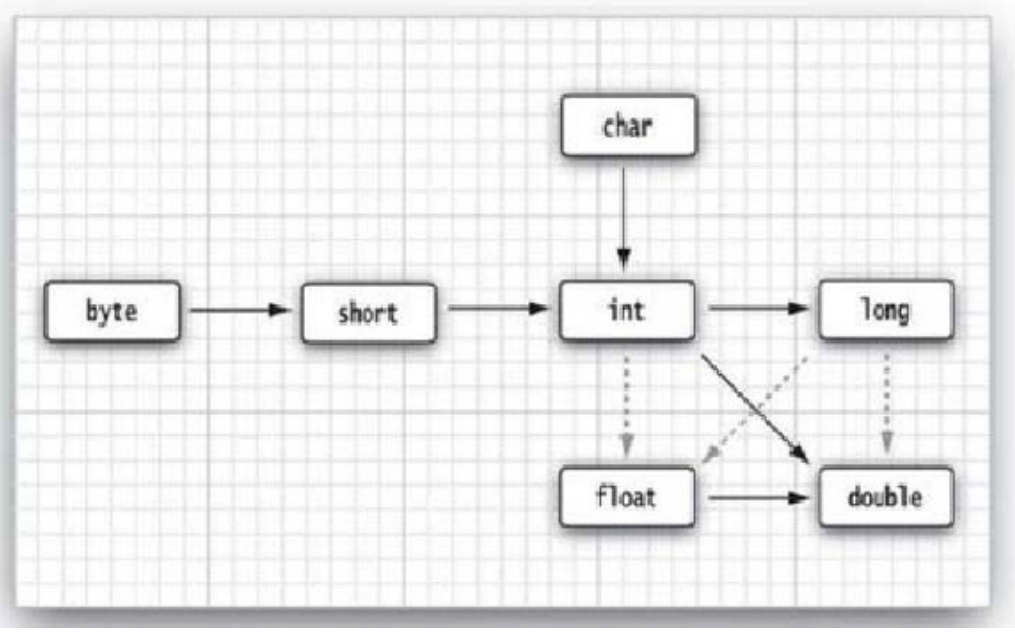
Literals文字

- long, float数字末尾加L，f
- null一般用于测试空值存在
- 下划线使用
  - 数字中，数字之间任何位置都可出现 5_____2
  - 不可出现情况
    - 数字开头或结尾 52_, 0x_52
    - 浮点数的小数点前后 3_.1415
    - F或L后缀之前　999_L
    - 需要一串数字的地方

```
// Literals
long l1 = 123L;
float f2 = 1.23f;
double d3 = 123.4;
double d4 = 1.234e2; // 等于d3
// Unicode转义序列可在char，String，字段名称等地方使用
String s = "S\u00ED Se\u00F1or" (Sí Señor in Spanish).
```

数值间类型转换

- 二元运算符两端操作数类型不一致，转换顺序 double > float > long > int
- 虚线表示可能会失去精度，int最多10位有效数字，double为15~16位



数组 array

- 定义 int[] ages; 数组类型+数组名
- 初始化
  - **数组创建后长度固定**
  - int[] anArray = new int[10];
  - anArray[0] = 100;
  - int[] anArray = {100,200,300};
- 多维数组
  - 多为数组本身组件也是数组

    - 允许各数组长度不同
- 函数
    - 数组长度 anArray.length
    - 数组拷贝 System.arraycopy(copyFrom, 2, copyTo, 0, 7) 拷贝7个字符，从copyFrom[2]到copyTo[0]
    - binarySearch() 搜索索引
    - equals() 数组元素相等，按序
    - fill()
    - sort() 升序排列

运算符

- + 加法运算符

String firstString = "This is";

String secondString = "an apple";

String thirdString = firstString + secondString; //"This is an apple"

- instanceof
    - 测试对象是特定接口/类的实例?
    - null不是任何实例
- >>>无符号右移

控制流语句

- **决策语句** decision-making statements
    - if-then, if-then-else, switch
    - switch开关：byte, short, char, int, 也适用于**枚举类**，**String类**，包含基本类的特殊类：Character, Byte, Short, Interger
- **循环语句** looping statements
    - for, while, do-while
    - for循环 for(int item : numbers){ }
- **分支语句** branching statements
    - break, continue, return

---

4 类与对象

类定义

- body 类体
  - field　域
  - constructor　构造函数
    - 没有返回值
    - 与类名相同
    - 无构造函数，编译器会提供一个默认的构造函数
  - method declarations　方法声明
- 继承：继承1个类extends，多个接口implements

方法定义

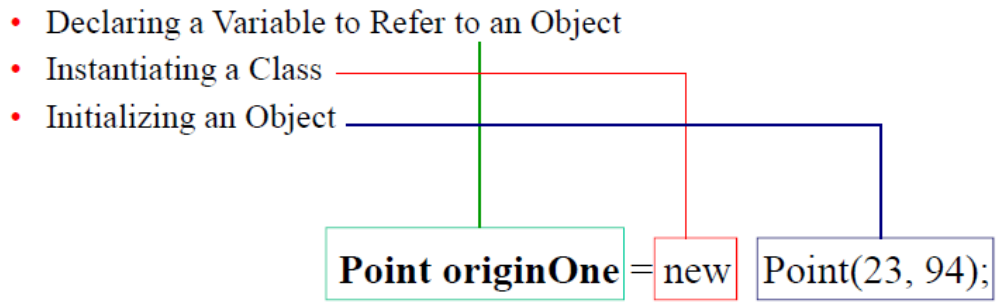
- 类名首字母大写，方法名第一个词应为动词 runFast, isEmpty
- 方法重载
  - 条件
    - 名称相同，参数数量 / 参数类型不同
    - **返回类型不同，不会引起重载**
  - 缺点
    - 重载会降低代码的可读性
- 参数类型
  - Java不允许将方法传递给方法，可以传递对象，然后调用该对象的方法
  - 传递任意数量的参数，可用varargs(三个点+空格)，**传入0或多个参数**

public Polygon polygonFrom(Point... corners)

  - 成员方法若有同名参数，可以隐藏域，通常仅在构造方法和设置特定域的方法中使用

对象/类

- new运算符
  - 为新对象分配内存并返回内存引用
  - 调用对象的构造函数
  - 只需要一个后缀参数：对构造函数的调用



- 垃圾收集
  - Java提供garbage collector，定期自动释放不再引用的对象所用的内存
  - 设置保存引用的变量为null，来显示删除引用
- 参数返回
  - void， return num 会出现编译错误
  - 类名作为返回类型，返回值必须为**该类或者其子类**，不能返回父类
  - **接口名作为返回类型**，返回对象必须实现指定接口
- this关键字
  - 构造函数的参数隐藏了本地对象参数
  - this在构造函数中可调用另一个构造函数（显示构造函数）
- 修饰符
  - 类的修饰符：public，private（空）
  - 类成员的修饰符：
    - public,
    - private,
    - protected,
    - package-private
  - 使用对特定成员有意义的最严格的访问级别，private
  - 避免常量之外的公共字段，公共字段倾向于链接到特定的implement，并限制代码灵活性

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

类成员

- 类变量/修饰符
  - static修饰静态字段或类变量，用类名访问（推荐），或实例引用访问（不推荐）
  - final 修饰符指此字段的值不能更改
  - 类变量由所有实例共享

```
//类名访问
ClassName.methodName(args)
//无需创建实例，直接访问
static int numberOfBicycles = 0;
public static int getNumberOfBicycles() {
   return numberOfBicycles;
}
```

- 类方法
  - 类方法不能直接访问实例变量或实例方法，必须使用对象引用
  - 类方法不能使用this关键字，因为没有可引用的实例

内部类 nested classes

```
class OuterClass {      // 类
      static int outerVariable = 1;
       static class StaticNestedClass {      // 静态内部类
             int staticVariable = outerVariable;
             final int finalVariable = 10;
             class localClass {      // 局部类
                    int localClassVariable = finalVariable;
             }
       }
       class InnerClass {}
}
```

- 作为OuterClass的成员，内部类的修饰符有：public, private, protected, package-private
- 优点
  - 对仅在一个地方使用的类进行逻辑分组
  - 增加了封装
  - 代码更具有可读性和可维护性

- 静态内部类
  - 无权访问封闭类的其他成员
- 非静态内部类
  - 无法定义或声明任何静态成员
  - 可访问封闭类的其他成员（即使被声明为private）
  - 要实例化内部类，必须先实例化外部类

局部类

- 与内部类相似
- 局部类只能访问声明为final的局部变量
- 可以有静态成员（常量变量）：原始类型或String类型
  - static final String farewell = "Bye Bye";

匿名类

- 匿名类 = 表达式 + 声明
- 不能声明静态初始化，或成员接口
- 不能声明构造函数

```
public class AnonymousClasses {
      interface HelloWorld {
            public void greet();
       }
      public void sayHello() {
            // 匿名类
            HelloWorld englishGreeting = new HelloWorld() {
                  public void greet() { System.out.println("world");
                  }
             }
       }
}
```

枚举类型

- for - each 迭代枚举类型

---

5 包

- 包定义
  - 类和接口的组合（type类型：类，接口）
  - 提供访问保护和名称空间管理
  - 基本类在 java.lang
  - 读写类在 java.io
- 创建包
  - package graphics; 必须是源文件中的第一行，每个源文件只能有一个package语句
  - 单个源文件，只能有一个同名public类型
- 命名包
  - 包名称以小写形式编写，以避免与类或接口冲突
  - 反向Internet域名，开始软件包名称，com.example.region.mypackage
  - 非有效包名，数组开头，特殊字符，用下划线代替

Legalizing Package Names

Domain Name	Package Name Prefix
hyphenated-name.example.org	org.example.hyphenated_name
example.int	int_.example
123name.example.com	com.example._123name

- 引用包成员
  - Java编译器会自动导入
    - java.lang
    - 当前包
  - 以完全限定的名称引用该成员
    - graphics.Rectangle myRect = new graphics.Rectangle();
  - 导入软件包成员
    - import graphics.Rectangle;
    - import语句，在package语句之后，类型定义之前
  - 导入成员的整个软件包
    - import graphics.*;
    - * 只能用于指定包中的所有类，不能用于匹配包中类的子集
    - 比如，import graphics.A*; 会产生编译错误

- 包的层次结构
  - java.awt.color软件包，java.awt.font软件包和其他java.awt.xxxx软件包未包含在java.awt软件包中。
  - 必须同时导入两个包
- 名称歧义
  - 两个包中的成员共享名称，且都已导入，必须使用限定名称来引用成员 graphics.Rectangle rect;
- 静态导入声明
  - java.lang.Math 定义许多静态方法，cos, sin, tan, sqrt, max, min, exp
  - import static java.lang.Math.*;
- 管理源文件和类文件
  - <path_one>\sources\com\example\graphics\Rectangle.java
  - 只提供类文件地址，可有效保护源文件
  - <path_two>\classes\com\example\graphics\Rectangle.class
  - 必须设置CLASSPATH，以便编译器和JVM找到适合类型的 .class 文件

---

6 接口与继承

接口

- 接口定义
  - 接口是引用类型
  - 接口包含 常量，方法签名，默认方法，静态方法，嵌套类型
  - 方法主体仅适用于默认方法和静态方法
  - **接口不能实例化**，只能由类实现，或其他接口扩展
- 接口主体
  - 包含抽象方法，
  - 默认方法 default，
  - 静态方法 static
  - 接口可包含常量声明，**默认是public, static, final** 可以省略这些修饰符
- 默认方法
  - 带方法体
  - 扩展含默认方法的接口
    - 不提及默认方法，扩展接口可继承默认方法

        - 重新声明默认方法，使其抽象
        - 重新定义默认方法，将其覆盖

继承

- 定义
    - 没有任何显式父类，每个类隐式都为Object的子类
    - 子类从其父类继承所有成员（fields，methods，nested classes）
    - 构造函数不是成员，不会被子类继承，可以被子类调用
- 子类
    - 成员继承
        - 子类继承public，protected成员，无论位于哪个包
        - 如果子类和父类位于同一个包，还继承package-private成员
        - 子类不继承private成员
    - 隐藏 / 覆盖 hide / override
        - 隐藏父类field （即使类型不同）
        - 覆盖父类method
        - 隐藏父类static method
        - 隐藏父类private method，不覆盖
    - 成员访问
        - 子类构造函数用super或隐式调用父类构造函数
        - 子类可通过继承public，protected修饰的method，nested method来访问父类private成员

Defining a Method with the Same Signature as a Superclass's Method

	Superclass Instance Method	Superclass Static Method
Subclass Instance Method	Overrides	Generates a compile-time error
Subclass Static Method	Generates a compile-time error	Hides

- 对象类型
    - 对象转型
        - 向上转型
        - 向下转型

```
// 向上转型，隐式转型
Object obj = new MountainBike();
MountainBike myBike = obj; // 编译错误
```

`// 向下转型，显式转型`

`MountainBike myBike = (MountainBike)obj;`

    - 对象可具有多类型
        - 自己的类
        - 该类实现的所有接口的类
- 继承关系
    - 实例方法 优先于 接口默认方法
    - 多重继承接口，已被覆盖的方法将被忽略
    - 从类继承的实例方法 可覆盖抽象接口方法
    - 两个或以上接口的默认方法冲突，产生**编译错误**，必须显式重新supertype方法
    - 接口中的**静态方法 不会被继承**
- 修饰符
    - 子类覆盖方法的modifier的公开程序，必须≥父类，否则**编译错误**

多态

- 定义
    - 对象变量可以引用多种实际类型，称为多态
- 绑定
    - 静态绑定编译时决定，动态绑定运行时决定
    - 动态绑定
        - 取决隐式参数的实际类型
    - 静态绑定
        - private
        - static
        - final
        - constructor
- 子类
    - 构造函数
        - 调用父类的构造函数，必须在子类构造函数第一行
        - 构造函数链
    - Java中只有基础类型不是对象

函数

- public String toString()

    - 返回对象的字符串表示形式
    - 字符串+其他数据类型：toString()自动调用
    - 返回值：对象名+@+对象哈希码
    - 一般建议子类重写此方法
- equals()
    - 比较基本数据类型，数值相等
    - 比较对象，引用指向同一对象
    - 比较对象相等（包含相同信息），需重写此方法
- clone()
    - 拷贝后不指向同一对象
    - protected，不能从另一个类和包的Object内调用它
    - 需要现在子类中重写为public
    - 浅表拷贝（shadow copy），对象中有子对象，需要深度拷贝，不可变对象不需深度拷贝（immutable object），如字符串
    - 子类对象支持clone()操作
        - 实现Cloneable接口
        - 重写clone方法为public
        - 调用super.clone()
        - 处理CloneNotSupportedException异常
- finalize()
    - java有自动垃圾回收功能，不用重写finalize()
- final
    - final变量，不能改变
    - final方法，不能被子类重写
    - final类，不能被继承
- abstract
    - abstract类，
        - 不能被实例化，可以被子类化
        - 可声明非static，final的field，public，protected，private方法
        - 可实现部分interface
    - 带有abstract方法，class必须声明为abstract
    - 子类若没有实现全部抽象方法，必须声明为abstract
    - abstract一般不与interface一起使用，interface是隐式抽象的

7 数字与串

Numbers包

- 原因
  - 作为方法参数
  - 类定义常量，MIN_VALUE,MAX_VALUE
  - 值的类型转换，进制转换，转成字符串
- Numbers子类方法
  - byte byteValue()
  - int compareTo(Double anotherDouble)
  - boolean equals(Object obj)
  - static int parseInt() 返回整形
- printf() / format()
  - java.io 包含PrintStream类
  - print / println / format / printf

```
int i = 461012;
```

```
System.out.format("The value of i is: %d%n", i);
```

The printf and format Methods

Converter	Flag	Explanation
d		A decimal integer.
f		A float.
n		A new line character appropriate to the platform running the application. You should always use %n, rather than \n.
tB		A date & time conversion—locale-specific full name of month.
td, te		A date & time conversion—2-digit day of month. td has leading zeroes as needed, te does not.
ty, tY		A date & time conversion—ty = 2-digit year, tY = 4-digit year.
tl		A date & time conversion—hour in 12-hour clock.
tM		A date & time conversion—minutes in 2 digits, with leading zeroes as necessary.
tp		A date & time conversion—locale-specific am/pm (lower case).
tm		A date & time conversion—months in 2 digits, with leading zeroes as necessary.
tD		A date & time conversion—date as %tm%td%ty
	08	Eight characters in width, with leading zeroes as necessary.
	+	Includes sign, whether positive or negative.
	,	Includes locale-specific grouping characters.
	-	Left-justified..
	.3	Three places after decimal point.
	10.3	Ten characters in width, right justified, with three places after decimal point.

- Math类

- Math.E 自然对数的底数　Math.PI
- Math.random()
  - 0.0 <= number < 1.0
  - int number = (int) (Math.random() * 10);
  - 生成一系列随机数，创建java.util.Random对象

Basic Math Methods

Method	Description
double abs(double d) float abs(float f) int abs(int i) long abs(long lng)	Returns the absolute value of the argument.
double ceil(double d)	Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
double floor(double d)	Returns the largest integer that is less than or equal to the argument. Returned as a double.
double rint(double d)	Returns the integer that is closest in value to the argument. Returned as a double.
long round(double d) int round(float f)	Returns the closest long or int, as indicated by the method's return type, to the argument.
double min(double arg1, double arg2) float min(float arg1, float arg2) int min(int arg1, int arg2) long min(long arg1, long arg2)	Returns the smaller of the two arguments.
double max(double arg1, double arg2) float max(float arg1, float arg2) int max(int arg1, int arg2) long max(long arg1, long arg2)	Returns the larger of the two arguments.

Exponential and Logarithmic Methods

Method	Description
double exp(double d)	Returns the base of the natural logarithms, e, to the power of the argument.
double log(double d)	Returns the natural logarithm of the argument.
double pow(double base, double exponent)	Returns the value of the first argument raised to the power of the second argument.
double sqrt(double d)	Returns the square root of the argument.

Characters

- char
  - char uniChar = '\u03A9'; // omega
  - 自动打包：把char传入方法，编译器把char转为Characters
    - Character ch = 'a';
  - 转义字符：\'　\"　\\　　System.out.println("She said \"Hello!\" to me.");

- Strings
  - String类中修改字符串的方法，实际上创建并返回一个包含操作结果的新字符串
  - getChars() 字符串转换为字符数组
- concat()
  - "My name is ".concat("Rumplestiltskin");
  - "Hello," + " world" + "!"
  - 字符串跨多行，用+连接
- valueOf()
  - 字符串转数字
  - float a = Float.parseFloat(args[0]); 返回原始类型
  - float b = (Float.valueOf(args[1])).floatValue(); 返回对象
- 字符函数
  - String s1 = Integer.toString(i);
  - int dot = s.indexOf('.');
  - char aChar = s.charAt(9);
  - String trim()；删除前后空白
  - int compareToIgnoreCase(String str)；
  - boolean matches(String regex)； 测试此字符串是否与指定的正则表达式匹配
- StringBuilder
  - 可修改的String
  - 把大量String连起来，用StringBuilder更有效
  - StringBuilder(String s)；构建StringBuilder，并在后接16个空字符
  - append()
  - insert()
  - setLength()
  - 容量会自动增加
  - sb.reverse();
- StringBuffer
  - 与StringBuilder完全相同
  - 方法是同步的，线程安全

---

8 注解

- 用途
  - 编译器信息：使用注释来检测错误或禁止显示警告
  - 编译时和部署时处理：生成代码，XML文件
  - 运行时处理：检查注释
- 应用
  - Java SE8之后，支持重复注释
  - 可应用于类型的使用
    - myString = (@NonNull String) str;
  - 注释类型看起来像接口定义，@开头
  - @Documented对注释批注，要使注释出现在Javadoc生成的文件中
  - @Deprecated 注释已弃用
  - @Override 若无法覆盖父类方法，编译错误
  - @SuppressWarnings 禁止以其他方式生成警告
  - @SafeVarargs 代码不会对其varargs参数执行潜在的不安全操作。将抑制与varargs使用有关的未经检查的警告。
  - @Retention
    - 保留注释指定存储方式
    - RetentionPolicy.SOURCE –标记注释仅保留在源级别中，并且被编译器忽略。
    - RetentionPolicy.CLASS –标记的注释由编译器在编译时保留，但被Java虚拟机（JVM）忽略。
    - RetentionPolicy.RUNTIME-标记的注释由JVM保留，因此可以由运行时环境使用。
  - @Target
    - 批注标记了另一个批注，以限制该批注可以应用于哪种Java元素。
    - ElementType.CONSTRUCTOR可以应用于构造函数。
    - ElementType.FIELD可以应用于字段或属性
  - @Inherited
    - 可以从父类继承注解类型。 （默认情况下，这是不正确的。）
    - 当用户查询注释类型并且类没有该类型的注释时，将查询该类的超类作为注释类型。
    - 仅适用于类声明。

---

9 异常

- 异常类型

    - Error
        - 不能预料和恢复，编译时检查的异常
        - 不受Catch或Specify Requirement约束
        - 硬件或软件错误
        - 比如，打开文件失败
    - Exception：checked exception
        - 可以预料和恢复
        - 受Catch或Specify Requirement约束
        - 用户传入错误文件名，好的程序需要捕获错误并通知用户
    - RuntimeException
        - 不能预料和恢复
        - 不受Catch或Specify Requirement约束
        - 比如空指针传给构造函数，NullPointerException
    - 抛出异常 throwing an exception
        - 创建异常对象，交给运行系统
        - 搜索调用堆栈的顺序和方法调用顺序相反
        - 没找到合适的异常处理程序，运行系统终止
    - 异常处理器exception handler
        - 打印异常，恢复程序
- try{} catch{} finally{}
    - try block
        - 一行代码或整个需要checked exception的代码
        - **try语句至少包含一个catch或finally**
    - catch block
        - ExceptionType 必须时从Throwable类基础的类名
        - 可以执行错误恢复，提示用户做出决定或使用链接的异常将错误传播到更高级别的处理程序。
    - finally
        - try block退出，finally始终执行
        - 把资源放进finally block，可确保资源总是可恢复的，防止资源泄露
        - 将清理代码（cleanup code）放在finally block
- throws
    - throw语句仅需要一个参数，throwable对象

    - 引发unchecked exception必须包括throws语句
- exception 优点
    - 将Error-Handling代码与“Regular”代码分开
    - 将错误传播到调用堆栈中
    - 分组和区分错误类型

---

10 IO

---

11 泛型

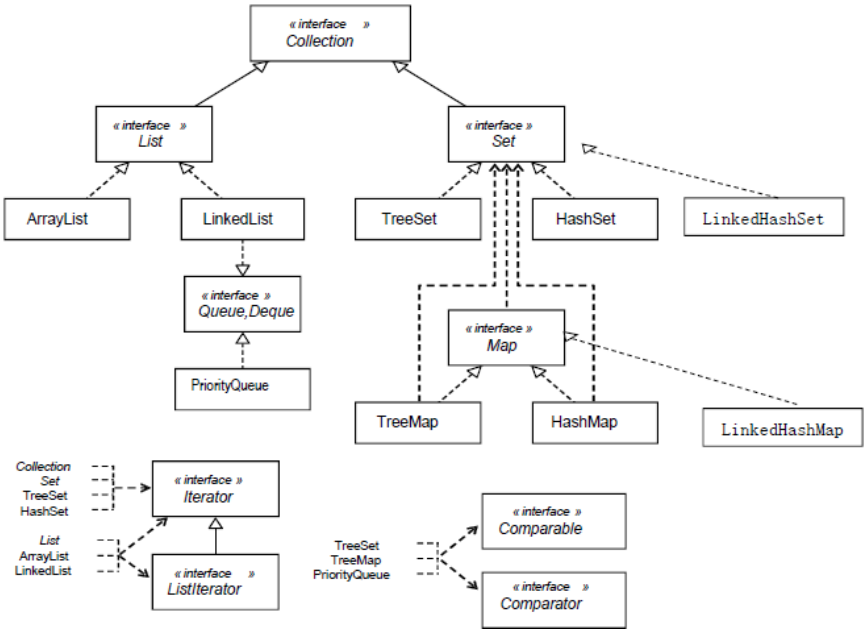
- 好处
    - 强类型，消除转型
    - 方法重用
- 旨在编译阶段有效
- 不能对确切的泛型类型使用instanceof操作。如下面的操作是非法的，编译时会出错。
    - if(ex_num instanceof Generic<Number>){ }
- 泛型类，是在实例化类的时候指明泛型的具体类型；
- 泛型方法，是在调用方法的时候指明泛型的具体类型 。

---

12 collection集合

- 集合定义
    - 接口：抽象数据类型。
    - 实现：可重用的数据结构。
    - 算法：多态的，可以在适当的许多不同实现中使用相同的方法。这些是对实现集合接口的对象执行有用的计算（例如搜索和排序）的方法。

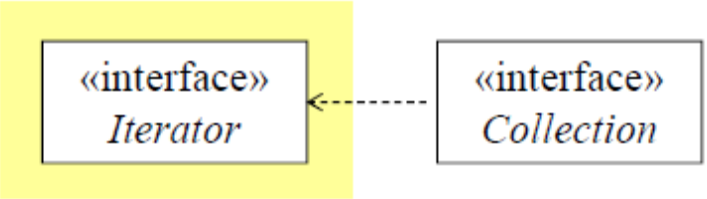
Collection Interface and Implementation



16

- 集合
  - Collection<E>
  - List<E>
  - Stack<E>
  - Set<E>
- 图
  - Map<K,V>
- 集合类型
  - Set
    - 不包含重复元素
    - TreeSet<E> Iterator按升序返回元素
    - HashSet<E> Iterator不按特定顺序返回元素
    - LinkedHashSet 链表运行的HashSet
  - List
    - 可包含重复元素
    - ArrayList 动态数组，空间不足会调整大小
    - LinkedList 带头节点的双向链表
    - 稳定排序（相同元素位置不变）
  - Queue
    -

        - 通常FIFO，优先队列除外
        - 每个Queue implement必须指定排序属性
    - Deque
        - 双端队列
    - Map
        - 不包含重复的键key
        - **几个键可以映射到相同的值**
        - TreeMap<K,V> 键集实现为二叉搜索树，O(log n)
        - HashMap<K,V> O(1)
        - LinkedHashMap
- Iterator



    - **每次调用next，只能调用一次remove**，否则引发异常
    - Iterator.remove是迭代过程中修改集合的唯一安全方法

---

13 多线程

- 线程
    - 定义
        - 可实例化线程
    - 中断
        - 线程通过在Thread对象上调用要被中断的线程来发送中断。
        - 线程长时间运行没有调用引发InterruptedException，必须定期调用Thread.interrupted()
        - 内部标志（称为中断状态）实现的。 调用Thread.interrupt设置此标志。
        - 当线程通过调用静态方法Thread.interrupted检查中断时，将清除中断状态。
        - 一个线程用于查询另一线程的中断状态的非静态isInterrupted方法不会更改中断状态标志。
        - 按照惯例，任何通过引发InterruptedException退出的方法都会清除中断状态。
    - join
        - join方法允许一个线程等待另一个线程的完成。

        - 如果t是当前正在执行线程的Thread对象，t.join（），导致当前线程暂停执行，直到t的线程终止。
    - 同步
        - 线程主要通过共享字段和对象引用字段进行通信，非常有效，但可能导致错误，线程干扰和内存一致性错误
        - synchronized(this)必须指定提供内部锁的对象

```
public synchronized void increment() {
c++;
}
synchronized(this) {
lastName = name;
nameCount++;
}
```

---

14 网络编程

- TCP
    - 当使用HTTP读取URL，必须按照发送顺序接收数据，否则将得到无效数据
    - URL，URLConnection，Socket和ServerSocket使用TCP网络通信
- UDP
    - 传递顺序不重要，不能保证，每个消息彼此独立
    - DatagramPacket, DatagramSocket, and MulticastSocket使用UDP通信
- 端口号
    - FTP 21
    - Telent 23
    - SMTP 25
    - DNS域名解析 53
    - HTTP 80
- URL
    - 协议标识符（http）+资源名称（example.com）：http://example.com
    -

```
// 方法1
```

```
URL myURL = new    URL("http://example.com/");
// 方法2
URL myURL = new    URL("http://example.com/pages/");
URL page1URL = new URL(myURL,    "page1.html");
// 方法3   http://example.com:80/pages/page1.html
URL gamelan = new URL("http",    "example.com", 80, "pages/page1.html");

// URL解析
import java.net.*;
import java.io.*;
public class ParseURL {
       public static void main(String[] args)    throws Exception {
              URL aURL = new    URL("http://example.com:80/docs/books/tutorial"
                     +    "/index.html?name=networking#DOWNLOADING");
              System.out.println("protocol = " +    aURL.getProtocol());
              System.out.println("authority = " +    aURL.getAuthority());
              System.out.println("host = " +    aURL.getHost());
              System.out.println("port = " +    aURL.getPort());
              System.out.println("path = " +    aURL.getPath());
              System.out.println("query = " +    aURL.getQuery());
              System.out.println("filename = " +    aURL.getFile());
              System.out.println("ref = " +    aURL.getRef());
       }
}
/* output
protocol = http
authority = example.com:80
host = example.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename =    /docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING
```

*/

---

15 正则表达式

- 字符匹配：可能要考

---

16 lambda表达式

- 匿名类
  - 同时声明和实例化一个类