《Java 软件开发基础》复习提纲

1. Java 语言的特点：面向对象、简单、平台无关性、健壮性和安全性、分布性、多线程、动态性、解释型
2. Java 程序的类型：Application、Applet、JSP/Servlet
3. 环境变量
   a) JAVA_HOME：JDK 的安装路径(C:\Program Files\Java\jdk1.7.0_04\)
   b) Path：Java 的编译器等 JDK 可执行文件的路径(C:\Program Files\Java\jdk1.7.0_04\bin)
   c) Classpath：JVM 运行时所需加载的类字节码文件的路径(.)
4. 编译和执行一个 Java 程序的命令
   a) Application
      i. javac HelloWorld.java
      ii. java HelloWorld
   b) Applet
      i. Javac HelloWorldApplet.java
      ii. appletviewer HelloWorldShow.html
5. Java 标识符、关键字
   a) 标识符
      i. 用来唯一地标识 Java 程序中的各种成分的名称，包括类名、对象名、方法名、常量名和变量名
      ii. 由字母、数字、下划线(_)和($)符号组成，但是标识符的第一个字符不允许为数字
      iii. 长度不限，区分大小写，为提高可读性应使用有意义的名称
   b) 关键字(保留字)
      i. Java 语言为特殊目的而保留的由 ASCII 字符构成的字符串
      ii. 关键字都有特定含义，只能用于特定位置，不能作为标识符使用
      iii. 有些关键字尚未使用，如 goto, const 等
      iv. 有些非关键字如 true 和 false 也不能作为标识符使用
6. 数据类型(基本类型和引用类型及区别)、类型转换
   a) 数据类型
      i. 确定了该类型数据的取值范围，由所占存储单元的多少而决定
      ii. 确定了允许对这些数据所进行的操作
      iii. 分类
         1. 基本数据类型
            a) 由 Java 语言定义，编译器可以理解，不需要外部的程序
            b) 占用内存的大小固定
            c) 数据存储在栈内存中
            d) 分类
               i. 整数类型：byte(1 Byte)、short(2 Byte)、int(4 Byte)、long(8 Byte)
               ii. 浮点类型：float(4 Byte)、double(8 Byte)
               iii. 字符类型：char(4 Byte)
               iv. 布尔类型：boolean(1 bit)

2. 引用数据类型
   a) 由类库开发者和程序员定义，占用内存的大小不固定
   b) 数据存储在堆内存中，通过栈内存中的引用类型变量访问
   c) 引用类型变量的值为数据在堆内存中的地址
   d) 分类：类(class)、接口(interface)、数组[]

b) 类型转换
   i. 隐式类型转换：byte → short → char → int → long → float → double
   ii. 显式类型转换：(<类型名>)<表达式>

7. 各种运算符、优先级、结合性
   a) 成员运算(.)、数组下标[]、括号()：自左至右
   b) 单目运算(++)(--)(-)(!)(~)：右/左
   c) 分配空间(new)、强制类型转换(type)：自右至左
   d) 算数乘(*)、除(/)、求余(%)运算：自左至右（左结合性）
   e) 算数加减运算(+)(-)：自左至右（左结合性）
   f) 位运算(<<)(>>)(>>>)：自左至右（左结合性）
   g) 小于(<)、小于等于(<=)、大于(>)、大于等于(>=)：自左至右（左结合性）
   h) 相等(==)、不等(!=)：自左至右（左结合性）
   i) 按位与(&)：自左至右（左结合性）
   j) 按位异或(^)：自左至右（左结合性）
   k) 按位或(|)：自左至右（左结合性）
   l) 逻辑与(&&)：自左至右（左结合性）
   m) 逻辑或(||)：自左至右（左结合性）
   n) 条件运算符(?:)：自右至左（右结合性）
   o) 赋值运算(=)(*=)(/=)(%=)(+=)(-=)(<<=)(>>=)(>>>=)(&=)(^=)(|=)：自右至左（右结合性）

8. 选择和循环语句、break 和 continue 语句
   a) 循环语句：基本同 C++
   b) 跳转语句
      i. 当程序中有嵌套的多层循环时，为从内循环直接退出外循环，可使用带标号的 break label 语句，此时应在要退出的外循环的入口语句前方加上 label 标号
      ii. 当程序中有嵌套的多层循环时，为从内循环跳到外循环，可使用带标号的 continue label 语句。此时应在外循环的入口语句前方加上 label 标号

9. 类与对象、对象的初始化、构造方法、生命周期
   a) 对象初始化的实现方法
      i. 用默认初始化原则赋初值
         1. 整数类型：0
         2. 浮点类型：0.0F
         3. 字符类型：‘\u0000’
         4. 引用类型：null
      ii. 由一些赋值语句赋初值
      iii. 使用构造方法
      iv. 使用初始化语句块（由编译器拷贝至每个构造方法中）

b) 构造方法：基本同 C++
   i. 如果一个用户在一个自定义类中未定义该类的构造方法，系统将为这个类定义一个无形式参数的默认构造方法
   ii. 如果构造方法调用了其他方法，被调用方法一般用 final 修饰以防止被子类覆盖
c) 使用对象：基本同 C++
d) 销毁对象
   i. Java 语言提供了内存垃圾的自动回收机制
      1. 在程序的运行时环境中，Java 虚拟机中存在垃圾回收器线程，负责自动回收没有用的对象占用的内存
      2. 减轻了程序员的负担，不会出现内存不足，内存错误释放和泄漏等问题，提高健壮性和稳定性
      3. 垃圾回收器作为低优先级线程进行垃圾回收。程序员可以调用 System.gc()通知垃圾回收器尽快进行垃圾回收，但垃圾回收器的运行时不可直接控制和不可预测的
   ii. 对象和内存垃圾：当没有引用变量只想某个对象时，该对象就会变成垃圾
   iii. Java 提供了一个名叫 finalize()的方法，用于在对象被垃圾回收机制销毁之前执行一些非内存资源的回收工作
      1. 由来及回收系统调用
      2. Finalize()方法没有任何参数和返回值
      3. 每个类有且只有一个 finalize()方法，由程序员实现
e) 对象生命周期：创建对象、使用对象、销毁对象

10. 继承的概念及作用
a) 定义：一个新类可以获得和使用一个已有类的所有非私有的变量和方法。在继承关系中，新类成为子类（派生类），而已有类称为父类（基类）
b) 特点
   i. 程序中父类对象出现的地方都可以用子类对象代替
   ii. 传递性
   iii. 能清晰体现相关类间的一般和特殊的层次结构关系。Java 语言中所有的类都是直接或间接地继承自 java.lang.Object
   iv. 实现代码复用
   v. 子类也可根据需要修改从父类继承的方法或增加新的方法
c) 方法重载(overload)：基本同 C++
d) 方法覆盖(override)
   i. 方法覆盖是指子类中一个成员方法的签名与父类中的一个成员方法的签名完全相同。方法的签名(signature)指方法的名称、参数列表和返回值类型
   ii. JDK5 之后允许子类方法的返回类型为父类方法返回类型的子类型，而不要求完全相同
   iii. 如果子类方法与父类方法的名称相同但参数列表不同，构成重载
   iv. 如果子类方法与父类方法的名称和参数列表相同，但修饰符不同，有可能导致编译错误
   v. 子类不能覆盖父类中用 final 或 static 或 private 修饰的方法
   vi. 覆盖体现了子类补充或改变父类方法的能力。通过覆盖，可以使一个方法在不同的子类中表现出不同的行为

Defining a Method with the Same Signatures as a Superclass' Method

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

11. 用于类、数据成员和成员方法的修饰符
    a) 类的修饰符
        i. public：可以被所有类访问。一个程序文件最多只能用一个类由 public 修饰，且这个文件的名称也必须与这个类的名称相同
        ii. 空(缺省)：该类可以被同一个包中的类访问
        iii. 包(package)：Java 类的组织单位。通常把功能相近的类置于同一个包中。同一个程序文件的所有类规定都在同一个包中的
        iv. protected 和 private：仅适用于内部类
        v. abstract：抽象类，包含至少一个抽象方法，其作用是提供一个共同的基类，为派生具体类奠定基础。抽象类没有完全实现，不能用于创建对象
        vi. final：该类不可以被其他类继承，适用于功能已经确定不需要扩充和修改的类。final 和 abstract 不能同时使用
    b) 成员变量修饰符
        i. static：静态变量属于类，被类的所有对象共享，相当于类中的全局变量，可以通过类名访问(类名.成员变量名)或(对象名.成员变量名)
        ii. 如果省略 static，则表示该成员变量为实例变量。实例变量属于一个具体对象，通过对象名访问
        iii. final：表明该成员变量为取值不会改变的常量
        iv. transient：表明该变量不参与对象序列化
        v. volatile：表明该变量可以被多个线程修改
    c) 成员方法修饰符
        i. static：静态方法
        ii. final：表明该方法不能被子类覆盖
        iii. abstract：抽象方法，只有方法声明，没有方法体
        iv. native：用其他程序设计语言（如 C 语言）实现的
        v. synchronized：同步方法，一次只让一个线程执行
        vi. throws：声明方法可能抛出异常
12. 数据成员、成员方法：基本同 C++
13. 局部变量和成员变量的生存期和作用域：基本同 C++
14. 类、数据成员和成员方法的访问控制
    a) public：所有类
    b) protected：同一个包中的类（含当前类）和所有子类
    c) 空(缺省)：同一个包中的类（含当前类）
    d) private：当前类
15. 多态的概念、方法重载和覆盖：基本同 C++

16. 动态绑定和静态绑定
    a) 每个类有一个方法表，表中每项对应于一个方法，索引到方法实现代码上。如果子类覆盖了父类中某个方法的代码，则该方法对应的表项的索引更换到子类的实现代码上，而不在方法表中出现新的项
    b) 动态绑定：如果该方法是实例方法（即方法声明没有 static 修饰符），则实际调用的方法必须在运行时间确定
        i. 在引用变量所引用的对象实际所属的类(类 A)的方法表中找到调用方法的对应的表项，由此找到调用方法的代码。该调用方法可能在类 A 中定义，也可能在类 A 的某个父类(类 B)中定义
        ii. 类 B 可以等于或者不等于引用变量的声明类型(类 C)
    c) 静态绑定：如果该方法是静态方法(即方法声明有 static 修饰符)，则实际调用的方法可以在编译时间确定，即静态绑定——只在引用变量的声明类型(类 C)中查找这个方法
17. this, super 的用法、调用父类的构造方法
    a) this 关键字：对当前对象的引用
        i. 用来访问当前对象的成员变量
        ii. 用来访问当前对象的成员方法
        iii. 用来调用同一个类中的其他构造方法
    b) super 关键字：对当前对象的直接父类对象的引用
        i. 用来访问直接父类隐藏的成员变量
        ii. 用来调用直接父类中被覆盖的成员方法
        iii. 用来调用直接父类的构造方法
18. 构造方法的继承和重载
    a) 如果一个类没有定义构造方法，系统将为这个类定义一个默认构造方法，并在构造方法加入一条语句 super();。在创建新对象时，它将执行继承自父类的无参数构造方法。如果父类没有定义无参数构造方法，会导致编译错误
    b) 如果子类自己定义了构造方法，则该方法可以：
        i. 调用本类中重载的构造方法
        ii. 调用父类的构造方法
        iii. 如果上述两种情况都没有发生，则编译器会在构造方法加入 super();作为第一条语句。在创建新对象时，它将先执行继承自父类的无参数构造方法，然后再执行自己的构造方法
    c) 对于父类的含参数构造方法，子类可以通过在自己的构造方法中使用 super 关键字来调用它，但这个调用语句必须是子类构造方法的第一个可执行语句
19. 继承的使用原则
    a) 子类能够继承父类中被声明为 public 和 protected 的成员变量和成员方法，但不能继承被声明为 private 的成员变量和成员方法
    b) 子类能够继承在同一个包中父类的由默认修饰符修饰的成员变量和成员方法
    c) 如果子类声明了一个与父类的成员变量同名的成员变量，则子类不能继承父类的成员变量，此时称子类的成员变量**隐藏**了父类的成员变量
    d) 如果一个子类定义了一个成员方法与父类的某个方法的声明（包括方法名、参数列表、返回值类型）完全相同，则子类不能继承父类的成员方法，此时称子类的成员方法**覆盖**了父类的成员方法

20. 抽象(abstract)类和抽象方法
    a) 抽象类的作用在于提供一个公共的父类将许多有关的类组织在一起，那些被他组织在一起的类作为它的子类由它派生出来
    b) 抽象类刻画了这些类的共有特征，并通过继承机制传送给她的派生类
    c) 凡是用 abstract 修饰符修饰的成员方法称为抽象方法。这些方法只有声明，没有方法体
    d) 在各子类继承了父类的抽象方法之后，再根据子类的需要分别实现方法体
21. 接口(interface)
    a) 实现某一特定功能所需的常量和抽象方法的声明的集合
    b) 这个功能是由在类声明中用关键字 implements 实现这个接口的各个类中完成的，即要由这些类来具体实现接口中各抽象方法的方法体
    c) 语法格式

```
[修饰符] interface 接口名 [extends 父接口名列表] {
    [public] [static] [final] 常量;
    [public] [abstract] 方法;
}
```

        i. 修饰符：可选，用于指定接口的访问权限
            1. 如果用 public 修饰，表明可以被所有类和接口访问
            2. 如果省略则使用默认的访问权限，可被同一个包内的类和接口访问
        ii. 接口名：用语指定接口的名称，必须是合法的 Java 标识符。一般情况下，要求首字母大写
        iii. extends 父接口名列表：可选，用于指定要定义的接口继承于哪些父接口，用逗号分隔
    d) 定义接口要注意
        i. 变量都是常量
        ii. 成员方法都是抽象方法
        iii. 接口也具有继承性，可以通过 extends 关键字声明该接口的父接口
    e) 在用类实现接口时要注意
        i. 用 implements 关键字指定这个类实现哪些接口，用逗号隔开
        ii. 如果类不是抽象类，则在累的定义部分必须实现指定接口的所有抽象方法，即为所有抽象方法定义方法体
        iii. 如果类是抽象类，则它可以不必实现接口所有的方法。但是对于这个抽象类的任何一个非抽象的子类而言，它们的父类所实现的接口中的所有抽象方法都必须有方法体。这些方法体可以来自抽象的父类，也可以来自子类自身，但是不允许存在未被实现的接口方法。这主要体现了非抽象类中不能存在抽象方法的原则
        iv. 接口的抽象方法的访问控制都已指定为 public，所以类在实现方法时，必须显式地使用 public 修饰符，否则将被系统警告缩小了接口中定义的方法的访问控制范围

f) instanceof 运算符
   i. 使用方法：对象名 instanceof 类名(接口名)
   ii. 返回值为布尔类型
      1. 结果为 true
         a) 左边对象是右边类的一个对象
         b) 左边对象是右边类的一个子类的对象
         c) 左边对象的类实现了右边的接口
      2. 结果为 false：其他情况

22. 包(Package)：一组功能相关的类和接口的集合
   a) Java 中提供的包主要用途
      i. 类的组织方式：将功能相近的类放在同一个包中，可以方便查找与使用
         1. java.lang 包含了 Java 语言的核心类和基本类，如 Object, Math, String, Thread 等
         2. java.awt 和 javax.swing 中包括了用来构建图形用户界面(GUI)的类
         3. java.io 支持输入/输出
         4. java.net 支持基于 TCP/IP 的网络通信
      ii. 提供了一种管理类的命名空间的机制。由于在不同包中可以存在同名类，所以使用包在一定程度上可以避免命名冲突
      iii. Java 中访问权限是以包为单位的
         1. 若类的声明有 public 修饰符，则表明该类可被所有类访问
         2. 若类的声明无 public 修饰符，则表明该类仅供同一包的类访问
   b) 创建包
      i. 语法格式：package 包名;
      ii. 包名：用于指定包的名称。包的名称必须为合法的 Java 标识符。当包中还有子包时，可以使用”包 1.包 2. … .包 n”进行指定，其中包 1 为最外层的包，而包 n 则为最内层的包
      iii. 作用是将本源文件中的接口和类纳入指定包。源文件中若有 package 说明语句，必须是第一个语句
      iv. 创建包就是在当前目录创建一个子目录层次结构，以便存放这个包中包含的所有类的.class 文件。定义多级包的分隔符’.’对应目录分隔符。文件目录结构必须匹配包的层次结构
      v. Java 虚拟机访问包的所有路径是由 Classpath 环境变量保存。如果程序运行时找不到类的错误(NoClassDefFoundError)，需要将类文件层次结构的根目录加到 Classpath 中
   c) 使用包
      i. 使用长名引用包中的类
      ii. 使用 import 语句引入包中的类

23. String 类常用方法
    a) 字符串：Java 语言规定字符串常量必须用双引号括起，一个串可以包含字母、数字和各种特殊字符
    b) String 类是 Java 类库中的常用类，Java 的任何字符串常量都是 String 类的对象
    c) 创建 String 类的对象
        i. String c1 = new String("Java"); // 直接再堆中创建一个对象
        ii. String c1 = "Java"; // 先在字符串池中查找，没有再在堆中创建
        iii. 用对象名.inter()可以强制检查：String c2 = c1.intern();
        iv. 其他常用构造方法
            1. String(byte[] bytes, int offset, int length)
            2. String(char[] value)
            3. String(char[] value, int offset, int count)
    d) String 类常用方法
        i. int length()：返回此字符串的长度
        ii. char charAt(int index)：返回指定索引处的字符
        iii. String substring(int beginIndex, int endIndex)：提取子串
        iv. String substring(int beginIndex)：提取子串
        v. boolean startsWith(String prefix)：测试此字符串是否以指定的前缀开始
        vi. boolean startsWith(String prefix, int toffset)：测试此字符串从指定索引开始的子字符串是否以制定前缀开始
        vii. boolean endsWith(String suffix)：测试此字符串是否以指定的后缀结束
        viii. 查找
            1. int indexOf(int ch)：返回指定字符在此字符串中第一次出现处的索引
            2. int indexOf(int ch, int fromIndex)：返回在此字符串中第一次出现指定字符处的索引，从指定的索引开始搜索
            3. int indexOf(String str)：返回指定子字符串在此字符串中第一次出现处的索引
            4. int indexOf(String str, int fromIndex)：返回指定子字符串在此字符串中第一次出现处的索引，从指定的索引开始
            5. int lastIndexOf(int ch)：返回指定字符在此字符串中最后一次出现处的索引
            6. int lastIndexOf(int ch)：返回指定字符串在此字符串中最后一次出现处的索引，从指定的索引处开始反向搜索
            7. int lastIndexOf(String str)：返回指定子字符串在此字符串中最右边出现处的索引
            8. int lastIndexOf(String str, int fromIndex)：返回指定子字符串在此字符串中最后一次出现处的索引，从指定的索引开始反向搜索
        ix. String toLowerCase()：将此 String 中的所有字符都转换为小写
        x. String toUpperCase()：将此 String 中的所有字符都转换为大写
        xi. String replace(char oldChar, char newChar)：返回一个新的字符串，它是通过用 newChar 替代字符串中出现的所有 oldChar 得到的
        xii. String replaceAll(String regex, String replacement)：使用给定的 replacement 替换此字符串中所有匹配给定的正则表达式的子字符串
        xiii. String replaceFirst(String regex, String replacement)：使用给定的 replacement 替换此字符串中匹配给定的正则表达式的第一个子字符串

xiv. String trim()：返回字符串的副本，忽略前导空白和尾部空白
xv. boolean equals(Object anObject)：将此字符串与指定的对象比较
xvi. boolean equalsIgnoreCase(String anotherString)：将此 String 与另一个 String 比较，不考虑大小写
xvii. String intern()：返回字符串对象的规范化表示形式
xviii. boolean isEmpty()：当且仅当 length()为 0 时返回 true

e) 类型转换
   i. 其他类型转为 String
      1. 基本数据类型转为 String：static String valueOf(Object obj)
      2. 基本数据类型的包装类转为 String：static String toString(Object obj)
      3. 其他类转为 String：static String toString()
   ii. String 转为其他类型
      1. 对于基本类型，可以使用对应的包装类的 parseXXX()方法
         a) static int parseInt(String s, int radix)
         b) static int parseInt(String s)
         c) static double parseDouble(String s)
      2. 可以使用基本类型的包装类的 valueOf()方法得到包装类
         a) static Double valueOf(String s)
         b) static Integer valueOf(String s)
         c) static Integer valueOf(String s, int radix)

f) 字符串连接
   i. 使用'+'操作符
   ii. 使用 String.concat()方法

g) 其他方法
   i. public int compareTo(String anotherString)：按字典顺序基于字符串中的各个字符的 Unicode 值比较两个字符串
   ii. boolean regionMathes(boolean ignoreCase, int toffset, String other, int ooffset, int len)：测试两个字符串区域是否相等
   iii. static String format(String format, Object … args)：使用指定的格式字符串和参数返回一个格式化字符串
   iv. void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)：将字符从此字符串复制到目标字符数组
   v. static String copyValueOf(char[] data, int offset, int count)：返回指定数组中表示该字符序列的 String

24. main 方法的命令行参数：基本同 C++