2023年JAVA考试速通小典

写在前面

本小典以王亮明老师所发的PPT以及最后一节课的划重点为基础进行整理，适合要求不太高没时间复习的同学。

整理者val，欢迎大家指正错误。

本小典仅供学习交流。

2023年12月Java考试

考试范围：

1234567，**9，10，11，12，13**，14，15

考试题型：

试卷近三年低于5%查重率。 其他两个老师出，王亮明老师不出。两张卷子，题型有差别但是基本一致。

- 选择题：15道30分，覆盖范围很广。
- 判断题：单词量
- 简答题：读程序写结果
- 编程大题

各类题型，有老师合起来，有老师分开。

- 30 + 5 + 5×3 + 5×4 + 30。
- 30 + 10 + 3×10 + 30。

划重点

- 基础java语法
- 面向对象
- 异常处理必考非常重要
- javafx一道基础的，主要考察事件相关的。UI里的组件如何读到数据？数据怎么处理？事件绑定。一道题，不多不难。
- 编程题文本文件读写相关必考，看课件/课本？例子。
- 字符串常规处理的方法，比如替换或者其他的，意思就是不考替换。跟其他数据的一些转换。注意一下看一下。
- 创建文件流、异常处理、循环读写这些会了就一半分
- 接口，cloneable和comparable接口，这两个接口。
- 第二第三个实验，用到的核心技术，考试都会涉及到。例如单例设计模式和接口的使用，异常处理。深浅拷贝等。如何实现？
- 自定义异常
- java跟C++差别大的多关注
- ascii码值可能会用到，出题老师觉得需要记一下

chapter 1

PPP面向过程编程 vs OOP面向对象编程

Advantages of oop

- modularity:
- information-hiding:
- code reusing:
- pluggability and debugging ease:

JVM、JRE、JDK

- JVM: Java Virtual Machine
- JRE: Java Runtime Environment
- JDK: Java Development Kit

How to run a java program

- create source code(eg.welcome.java)
- compile source code(javac welcome.java)->bytecode(eg.welcome.class)
- run bytecode（eg.java welcome）
- result:"welcome to java!"

Platform-independent program

the result of a compiled java technology rogram is platform-independent bytecodes,which can be executed by any system that implements the java virtual machine.

Anatomy of a java program

- class name
- main method
- statements
- statement terminator
- reserved words
- comments
- blocks

chapter 4 mathematical functions, characters, and strings

ASCII code

8位编码表。Unicode码包含ASCII码，其中‘u0000‘to‘u007F‘是128个ASCII码。 一些常用字符的 ASCII 码值如下所示：

- '0'-'9':48-57 \u0030-\u0039
- 'A'-'Z':65-90 \u0041-\u005A
- 'a'-'z':97-122 \u0061-\u007A

casting between char and numeric types

char类型数据可以转换成任意一种数值类型，反之亦然。将整数转换为char类型数据的时候，**只用到该数据的低16位**，其余部分被忽略

the string type

the string type is not a primitiv type. it is known as a reference type(引用类型).

any java class can be used as a reference type for a variable.

例如，String message = “Hello Java”，引用类型定义了引用变量，变量引用了内容为`halo java`的字符串对象

simple methods for string objects

- length()
- charAt(int index)
- toLowerCase()
- toUpperCase()
- trim():return a new string with the leading and trailing spaces removed?

instance method and static method

- a static method can be invoked without using an object.all the methodsdefined in the math class are static methods.
- an instance method can be invoked only through an object of the class in which the method is defined.
- 实例方法可以调用实例方法和静态方法，以及访问实例数据域和静态数据域；
- 静态方法只能调用静态方法和访问静态数据域，不能调用实例方法和访问实例数据域。

comparing strings

字符串比较：操作符`==`只能检测两个字符串是否存储在同一个位置（指向同一个对象），而不能检测两个字符串是否相等。要检测两个字符串是否相等，必须使用`equals`或者`comparableTo`方法。

- comparableTo方法返回一个整数，如果字符串相等，返回0；如果字符串不相等，返回一个s1和s2从左到右第一个不同字符之间的ASCII码差值。
- 如果字符串相等，equals方法返回true；如果字符串不相等，equals方法返回false。

conversion between strings and numbers

- 可以将数值型字符串转换为数值，使用`Integer.parseInt()`、`Double.parseDouble()`等方法。如果字符串不能转换为数值，会抛出NumberFormatException异常。
- 将数值转换为字符串，只需要使用字符串连接操作符+即可。例如，`String s = "" + 5`;将整数5转换为字符串"5"。或者`string s1 = String.valueOf(s)`；也可以使用`Integer.toString()`、`Double.toString()`等方法将数值转换为字符串。

chaper 7 single-dimensional arrays

the Array.sort method

- `java.util.Arrays.sort(array)`
- `java.util.Arrays.sort(array,fromIndex,toIndex)`

chapter 9 objects and classes

常量应该被声明为`final static`

array of objects

array of objects is an array of object references variables. it is not an array of objects. `circle[] circleArray = new circle[10]`

immutable objects and classes不可变对象和类

if the contents of an objects cannot be changed after it is created, the object is called an immutable object. the class of an immutable object is called an immutable class.eg.String类

what class is immutable?

for a class to be immutable, it must mark all data fields private and provide no mutator methods(no accessor methods that would return a reference to a mutable data field object). (提供一个`get`方法返回数据域的值；提供一个`set`方法为数据域设置新值。这两者称为`accessor mathods` 和 `mutator methods`)

chapter 10

wrapper classes包装类

- Java中许多方法需要将对象作为参数，可使用java api中的包装类将基本数据结构类型包装成对象。例如将int类型的数据包装成`Integer`对象，将double类型的数据包装成`Double`对象等。包装类的对象可以作为参数传递给方法，也可以作为方法的返回值。
- Java在`java.lang`包中提供了8个包装类，分别对应8种基本数据类型。这8个包装类分别是：`Byte`、`Short`、`Integer`、`Long`、`Float`、`Double`、`Character`和`Boolean`。
- 优势：使用包装类可将基本数据类型作为对象处理

automatic conversion between primitive types and wrapper classes types

- 自动装箱：将基本数据类型转换为包装类类型。例如，`Integer i = 5;`将整数5自动转换为Integer对象;`Integer[] intArray = {1, 2, 3, 4, 5};`将整数1~5自动转换为Integer对象并存储在数组中。
- 自动拆箱：将包装类类型转换为基本数据类型。例如，`int i = new Integer(5);`将Integer对象转换为整数5;`System.out.println(intArray[0]+intArray[1]);`将Integer对象转换为整数1和2，然后进行加法运算。

StringBuilder class and StringBuffer class

- Java中的String类是不可变类，即String类的对象一旦创建，就不能修改。如果需要修改字符串，只能创建一个新的字符串对象。这样会导致大量的字符串对象被创建，占用大量的内存空间，降低程序的执行效率。为了解决这个问题，Java提供了两个可变字符串类：StringBuilder和StringBuffer。这两个类的对象可以修改，而且修改后不会创建新的对象，因此效率更高。

```
StringBuilder sb = new StringBuilder("Welcome");
sb.append(" to Java");
sb.insert(0, "hello ");
sb.delete(8, 11);
sb.reverse();
System.out.println(sb);
```

> StringBuilder和StringBuffer类的对象都是可变的字符串序列，它们的方法基本相同，主要区别在于StringBuilder类的方法不是线程安全的，而StringBuffer类的方法是线程安全的。因此，如果需要在多线程环境下使用可变字符串，应该使用StringBuffer类。

chapter 11 inheritance and polymorphism

Superclass and Subclass

- A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

using the super keyword

the super keyword refers to the superclass of the class in which the keyword appears. it is used to (access superclass members,?) invoke superclass constructors, and invoke superclass methods that are overridden by subclass methods.

- 不同于于属性和普通方法，父类的构造方法不会被子类继承，他们只能使用关键字`super`来**调用父类的构造方法。**
- **super()必须是子类构造方法的第一条语句**，否则编译出错。这是显式调用父类构造方法的唯一方式，如果没有显式调用，系统会自动调用父类的无参构造方法。
- invoking a superclass constructor's name in a subclass **causes a syntax error**.

constructor chaining 构造方法链

这一块类比C++。

overriding methods in the superclass

overriding vs overloading

- 重载意味着使用同样的名字但不同的签名来定义多个方法
- 重写意味着在子类中提供一个对方法新的实现
- 方法重写发生在通过继承而相关的不同类中；方法重载可以发生在同一个类中也可以发生在继承的子类中

the object class and its methods

- every class in java is descended from the java.lang.Object class. if a class does not extend another class, it is implicitly a subclass of Object.

```
public class Circle {
    // ...
}
```

equivalent to

```
public class Circle extends Object {
    // ...
}
```

the toString method in the object class

- the toString method returns a string representation of an object. the default toString method in the Object class returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@`, and the unsigned hexadecimal representation of the hash code of the object.

polymorphism

dynamic binding

Which implementation is used will be determined dynamically by the Java Virtual Machine at runtime. This capability is known as dynamic binding.

If o invokes a method p, **the JVM searches the implementation for the method p** in C_1 , C_2 , ..., C_{n-1} and C_n (一系列层层继承的子类，C_1 辈分最小，o都是他们的实例对象) , in this order, **until it is found.**

method matching vs binding

两个不同的概念。

- 方法匹配：在编译时刻，编译器根据方法名和参数列表选择调用哪个方法。
- 方法绑定：在运行时刻，虚拟机根据对象的实际类型选择调用哪个方法。

casting objects

除了用casting operator来转换变量的基本数据类型，Casting can also be used to convert an object of one class type to another within an inheritance hierarchy.

- 一个对象可以被强制转换为它的**子类类型**，这称为向下转型。例如，`Object object = new Circle();`，将Circle对象转换为Object对象，这称为向上转型，这种转换也称为隐式转换。；
- `Circle circle = (Circle)object;`，将Object对象转换为Circle对象，这称为向下转型。

the instanceof operator

Use the instanceof operator to test whether an object is an instance of a class：

```
object o = new Circle();
if (o instanceof Circle) {
    System.out.println("The circle diameter is " + ((Circle) o).getDiameter());
    // ...
}
```

the ArrayList class

Java provides the ArrayList class that can be used to **store an unlimited number of objects.**

存储在ArrayList中的元素**必须是一种对象**，不能使用诸如int的基本数据类型来代替一个泛型类型。但是，可以创建一个存储Integer对象的ArrayList

访问权限summary

Modifier on members in a class	Access from the same class	Access from the same package	Access from a subclass	Access from a different package
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default	Yes	Yes	No	No
private	Yes	No	No	No

chapter 12 Exception handling and text I/O

Exception handling

checked exceptions vs. unchecked exceptions

- RuntimeExeption、Error and their subclasses are unchecked exceptions（免检异常）.
- All other exceptions are checked exceptions（必检异常，meaning that thecompiler forces the programmer to check and deal with the exceptions，必检异常意味着编译器会强制程序员检查并通过try-catch块处理它们，或者在**方法头**进行声明。）

declaring exceptions

Every method must state the types of checked exceptions it might throw. This is known as declaring exceptions.

```
public void myMethod() throws IOException,
OtherException1, …, OtherExceptionN
```

Throwing exceptions

try-throw-catch

在执行完catch块之后，程序控制不返回到throw语句；而是执行catch块之后的下一条语句。

这一块感觉也和C++差不多，不放心可以自己再看看课件。

Rethrowing Exceptions

如果异常处理器不能处理一个异常，或者只是简单地想让调用者注意到该异常，JAVA 允许该异常处理器重新抛出异常。

```
try {
statements;
}
catch(TheException ex) {
perform operations before exits;
throw ex;
}
```

语句 `throw ex` 重新抛出异常给调用者，以便调用者的其他处理器获得处理异常`ex`

The finally Clause

无论异常是否产生，finally 子句总会被执行

```
try {
statements;
}
catch(TheException ex) {
handling ex;
}
finally {
finalStatements;
}
```

Defining Custom Exception Classes

```
public class MyException extends Exception {
public MyException() {
super();
}
public MyException(String message) {
super(message);
}
}
```

Text IO

> A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file.
>
> In order to perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file.
>
> This section introduces how to read/write strings and numeric values from/to a text file using the Scanner（读数据）and PrintWriter classes （写数据）.

printwriter

调用printWriter的构造方法创建一个新文件，调用该构造方法可能会抛出一个I/O异常，Java强制要求要编写代码来处理这类异常，因此，简单起见只需在方法头声明中申明`throws IOException`（第2行）

```
public class WriteData {
public static void main(String[] args) throws java.io.IOException {
// Create a File object
File file = new File("scores.txt");
// Create a PrintWriter object
PrintWriter output = new PrintWriter(file);
// Write formatted output to the file
output.print("John T Smith ");
output.println(90);
output.print("Eric K Jones ");
output.println(85);
// Close the file,必须使用close()方法关闭文件。如果忘记写，数据就不能正确的保存在文件中
output.close();
}
}
```

Try-with-resources

Programmers often forget to close the file. JDK 7 provides the followings new try-with-resources syntax **that automatically closes the files**.

```
try (declare and create resources) {
Use the resource to process the file;
}
```

chapter 13 abstract classes and interfaces

abstract classes and abstract methods

略。

Interfaces vs. Abstract Classes

在 Java 中，接口被视为特殊的类。 每个接口都被编译成一个单独的字节码文件，就像常规类一样。 与抽象类一样，您无法使用 new 运算符从接口创建实例，但在大多数情况下，您可以以与使用抽象类大致相同的方式使用接口。 例如，您可以使用接口作为变量的数据类型、转换的结果等。

- In an interface, the data must be **constants**; an abstract class can have **all types of data**.
- Each method in an interface **has only a signature without implementation**; an abstract class **can have concrete methods**.

All classes share a single root, the Object class, **but there is no single root for interfaces**. Like a class, an interface also defines a type. A variable of an interface type can reference any instance of the class that implements the interface. **If a class extends an interface, this interface plays the same role as a superclass**. You can use an interface as a data type of an interface type to its subclass, and vice versa. and cast a variable

define an interface定义接口

To distinguish an interface from a class, Java uses the following syntax to define an interface:

```
public interface InterfaceName {
constant declarations;
abstract method signatures;
}
```

- 一个接口可以继承多个接口；例如：interface a extends cls1,cls2;
- 一个类可以实现多个接口；例如：class b implements face1 , face2 ；
- 一个类只能继承一个类，这就是JAVA的继承特点;

interfaces supports multiple inheritance

the comparable interface

Comparable 接口定义了compareTo 方法，用于比较对象

```
// This interface is defined in
// java.lang package
package java.lang;
public interface Comparable<E> {
public int compareTo(E o);
}
```

Comparable接口是一个范型接口，在实现该接口时，范型类型E被替换成一种具体的类型。compareTo 方法判断这个对象相对于给定对象o 的顺序，并且当这个对象小于、等于或大于给定对象o 时，分别返回负整数、0或者正整数。

Generic sort Method

如果对象是Comparable接口的实例，Java API中的java.util.Arrays.sort(Object[])方法可以使用compareTo方法对数组中的对象进行比较和排序。

the cloneable interface

Cloneable Interface给出了一个可克隆的对象。经常会出现需要创建一个对象拷贝的情况。为了实现这个目的，需要使用 clone 方法并理解 Cloneable接口。 接口包括常量和抽象方法，但是 Cloneable 接口是一个特殊情况，它是空的。

> **一个方法体为空的接口称为标记接口（marker interface）**。它用来表示一个类拥有某些希望具有的特征。

A class that implements the Cloneable interface is marked cloneable, and its objects can be cloned using the clone() method defined in the Object class.（实现Cloneable接口的类标记为可克隆的，而且它的对象可以使用在*object类中定义的clone()方法*克隆）

```
package java.lang;
public interface Cloneable {
}
Object类定义了一个clone方法，其方法头是：

```java
protected native Object clone() throws
CloneNotSupportedException;
```

- 关键字native表示这个方法不是用Java写的，但它是在JVM中针对 本地平台实现的。
- 关键字protected限定方法**只能在同一个包中或其子类中访问。***通常其他类必须重写该方法并将它的可见性修改为public，这样重写的clone方法可以在任何一个包中使用*
- Object类的clone方法完成了克隆对象的任务，实现浅拷贝：将原始对象的每个数据域复制给目标对象。如果一个数据域是基本类型，复制的就是它的值；如果一个数据域是对象，复制的就是其引用。。
- 所以综上，一个对象能被克隆，通常需满足如下2个条件

1. 实现Cloneable接口，使拷贝合法，避免抛出CloneNotSupportedException
2. 重写clone方法

深浅拷贝

- Shallow Copy（浅拷贝）是指拷贝对象时仅仅拷贝对象本身，包括对象中的基本变量，而不拷贝对象包含的引用指向的对象。Object类定义的clone方法为浅拷贝。最终拷贝出来，原始对象与拷贝对象共享对象包含的引用指向的对象。
- Deep Copy（深拷贝）不仅拷贝对象本身，而且拷贝对象包含的引用指向的所有对象。最终拷贝出来的对象与原始对象没有任何关联，变量中的对象是独立的，基本变量是相同的。深拷贝需要自己实现。

chapter 14 JavaFX basics

basic strucure of JavaFX
```

- Application（每个JavaFX应用程序定义在一个继承自`javafx.application.Application`的类中）
- Override the start(Stage) method（重写javafx.application.Application类start方法）
- Stage, Scene, and Nodes
    - Stage（舞台）对象是**一个窗体，它用于显示场景**。当应用程序启动的时候，一个称为主舞台的对象（`primaryStage`）由**JVM自动创建并传递给Start方法**
    - Scene（场景）对象可以在Stage中展示。一个Scene对象可以使用构造方法`Scene(node,width,height)`创建，该方法指定场景的宽度和高度，并将结点置于场景中。
    - Nodes（结点）：面板、组、控件以及形状都是结点。Button（按钮）对象是结点的一种，可以置于一个Scene对象中

```
import javafx.application.Application;//该类定义了JavaFX应用程序的基本结构
import javafx.scene.Scene;//该类是JavaFX应用程序的场景类
import javafx.scene.control.Button;//该类是JavaFX应用程序的按钮类
import javafx.stage.Stage;//该类是JavaFX应用程序的舞台类
public class MyJavaFX extends Application {
@Override // Override the start method in the Application class
public void start(Stage primaryStage) {
// Create a scene and place a button in the scene
Button btOK = new Button("OK");
Scene scene = new Scene(btOK, 200, 250);
primaryStage.setTitle("MyJavaFX"); // Set the stage title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
}
public static void main(String[] args) {
Application.launch(args);//launch方法是一个定义在Application类中的静态方法，用于启动一个独立的JavaFX应用
}
}
```

在命令行中不需要main方法，直接运行`java MyJavaFX`即可。只有在JavaFX支持有限的IDE中才需要main方法。

VBox

就是一个垂直列节点布局控件。

- 对比FlowPane， FlowPane可以把子结点布局在多行或多列中，而Hbox或Vbox只能把子结点布局在一行或一列中。
- setMargin()方法用于将结点加入Vbox或其他布局控件的时候设置结点的外边距

chapter 15 event-driven programming and animation

Java Event Delegation Model(基于委派的模型)

Java采用基于委派的模型（the delegation model）进行事件的处理：源对象触发一个事件，然后一个对该事件感兴趣的对象处理它。后者称为事件处理器或事件监听器。

如果**一个对象**要成为源对象的事件处理器，需要满足如下2个条件：

1. 一个监听器对象是**对应的事件处理接口的实例**。JavaFX为事件T定义了一个统 一的处理器接口`EventHandler<T extends Event>`,该处理器接口包含`handle(T e)`方法 用于处理事件。
2. 该对象通过调用`source.setOnXEventType(listener)`进行事件处理器注册

Inner Classes

Inner class: A class is a member of another class.

- Advantages:
  - In some applications, you can use an inner class to make programs simple.
  - An inner class can reference the data and methods defined in the outer class in which it nests, so you do not need to pass the reference of the outer class to the constructor of the inner class.
  - Inner classes can make programs simple and concise.

An inner class supports the work of its containing outer class and is compiled into a class named `OuterClassName$InnerClassName.class`. For example, the inner class `InnerClass` in `OuterClass` is compiled into `OuterClass$InnerClass.class`.

Anonymous Inner Classes(匿名内部类)

Inner class listeners can be shortened using anonymous inner classes. An anonymous inner class is **an inner class without a name**. It **combines declaring an inner class and creating an instance of the class in one step.**

An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {
// Implement or override methods in superclass or interface
// Other methods if necessary
}
```

An anonymous inner class is compiled into a class named `OuterClassName$n.class`. For example, if the outer class Test has two anonymous inner classes, these two classes are compiled into `Test$1.class` and `Test$2.class`.

Simplifying Event Handing Using Lambda Expressions

Lambda expression is a new feature in Java 8. Lambda expressions can be viewed as an anonymous method with a concise syntax. For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

```
// (a) Without lambda expression
btOK.setOnAction(new EventHandler<ActionEvent>() {
@Override // Override the handle method
public void handle(ActionEvent e) {
System.out.println("OK button clicked");
}
});
```

```
// (b) With lambda expression
btOK.setOnAction(e -> System.out.println("OK button clicked"));
```

Basic Syntax for a Lambda Expression

The basic syntax for a lambda expression is either `(type1 param1, type2 param2, ...) -> expression` or `(type1 param1, type2 param2, ...) -> { statements; }`

The data type for a parameter may be explicitly declared or implicitly inferred by the compiler. The parentheses(圆括号) can be omitted if there is only one parameter without an explicit data type.

单例模式的实现

- 懒汉式，线程不安全

```
public class Singleton {
private static Singleton instance;
private Singleton() {
}
public static Singleton getInstance() {
if (instance == null)
instance = new Singleton();
return instance;
}
}
```

- 饿汉式，线程不安全

```
public class Singleton {
private static Singleton instance = new Singleton();
private Singleton() {
}
public static Singleton getInstance() {
return instance;
}
}
```

- 懒汉式，线程安全

```
public class Singleton {
private static Singleton instance;
private Singleton() {
}
public static synchronized Singleton getInstance() {
if (instance == null)
instance = new Singleton();
return instance;
```

```
    }
    }
```