1. Features of Java

   简单的（simple）

   面向对象的（object-oriented）

   分布式的（distributed）

   解析性的（Interpreted）

   健壮的（robust）

   安全的（secure）

   跨平台的，架构中性的（architecture neutral）

   可移植的（portable）

   高性能的（high performance）

   多线程的（multithreaded）

   动态语言（dynamic language）

2. A typical Java development environment

   Java 的程序正常执行五个步骤：编辑，编译，加载，验证，执行。

   Java programs normally undergo five phases: Edit, Compile, Load, Verify, Execute.

3. To test-drive a Java application using the command line

   编译 Java 时用 javac xxx.java 命令编译目录下全部文件 javac *.java 命令。

   运行 Java 时用 java xxx.class 命令 （.class 扩展名可以省略，即 java xxx）。

4. To Know some convention and syntax.

   给程序加注释（//注释内容或/*注释内容*/）（Commenting Your Programs）。

   声明类（每个程序至少包含一个类，类关键字后紧接着类名）（Declaring a Class）。

   类名和标识符（类名开头都用大写字母）（Class Names and Identifiers）。

   声明方法（类里包含一个或多个方法，java 程序里面必须包含 main 函数，执行应用程序必须调用 main 函数）（Declaring a Method）。

5. To write simple Java applications.

   main method

   public static void main(String[] args)

6. To use input and output statements.

   Class Scanner (nextInt …)没有 nextChar()

   OutputStream (System.out.println(…))

   \n 换行符　\t 水平制表符　　\r 回车符　\\输出’\’　\”输出”

7. Java’s primitive types

   boolean, byte, char, short, int, long, float, double

   　　1　　8　　16　　16　32　64　　32　　64

8. Basic memory concepts.

   Garbage collection

1）程序员无权回收内存。当一个对象没有被任何引用类型的变量引用时，该对象的内存可以被垃圾回收器自动回收；

2）垃圾回收器以独立的线程异步执行，回收的时间是程序无法预计的；

3）程序员可以调用 System.gc()或者 Runtime.gc()，提示垃圾回收器进行垃圾回收；

4）垃圾回收器进行垃圾回收时，调用对象的方法 finalize 析构方法。

5）Java 的垃圾回收机制是 Java 虚拟机提供的能力，用于在空闲时间以不定时的方式动态回收无任何引用的对象占据的内存空间。

考点：

1）注意条件，没有被任何引用类型的变量引用时

2）程序员无权强制回收，只能建议。

3）对象没有被回收的话就不会调用 finalize 方法

9. class and object

构造函数在类中没有显示地写出构造函数的时候默认，编译器会提供一个不带参数的默认构造函数，实例变量会被初始化成默认值。

10. instance variables and local variables

实例变量（instance variable）如果不初始的话将会有默认值 0 或 null，作用域是整个对象；局部变量（local variable）在方法

中声明，使用前必须初始化，否则报错，作用域只有在这个方法中。

11.primitive and reference types

1）基本数据类型（primitive type）：

boolean, byte, char, short, int, long, float, double

一次只能声明类型的一个值

2）引用数据类型（reference type）：

a.所有非基本数据类型均为引用数据类型（All nonprimitive types are reference types）；

b.对象（Objects）；

c.默认值为 null（Default value of null）

d.调用一个对象的方法（Used to invoke an object’s methods）

e. 引用数据类型在比较数据时应该使用 equals 方法，否则比较的是两个地址

primitive variable 是在栈中创建，一旦超出作用域将被销毁；

reference variable 的实例在栈中，保存的是数据在堆中的物理地址，只是一个“引用”。

12.Member access modifiers

1）public：被其修饰的类、属性以及方法不仅可以跨类访问，而且允许跨包访问。

2）private：被其修饰的类、属性以及方法只能被该类的对象，其子类不能访问，更不允许跨包访问。

3）protect：被其修饰的类、属性以及方法只能被类本身的方法以及子类访问。即使子类在不同的包中也可以访问。

4）default：该模式下，只允许在同一个包中进行访问

13.JOptionPane

1）JOptionPane.showMessageDialog(null, "友情提示");

2）JOptionPane.showMessageDialog(jPanel, "提示消息", "标题",JOptionPane.WARNING_MESSAGE);

3）JOptionPane.showMessageDialog(null, "提示消息.", "标题",JOptionPane.ERROR_MESSAGE);

4）JOptionPane.showInputDialog(null,"请输入你的爱好：\n","title",JOptionPane.PLAIN_MESSAGE,icon,null,"在这输入");

14.if and if else selection statements

if：如果条件为 true，就执行一种操作，如果是 false 就跳过（Single-selection statement）（Performs an action, if a condition is true; skips it, if false.）。

if…else：如果条件为 true，就执行一种操作，如果是 false 就执行另一种不同的操作（Double-selection statement）（Performs an action if a condition is true and performs a different action if the condition is false.）

15.？：

三元运算符（Ternary operator），操作数和？：形成条件表达式（Operands and ?: form a conditional expression）。

boolean 表达式：值（当表达式为 true）？值（当表达式为 false）

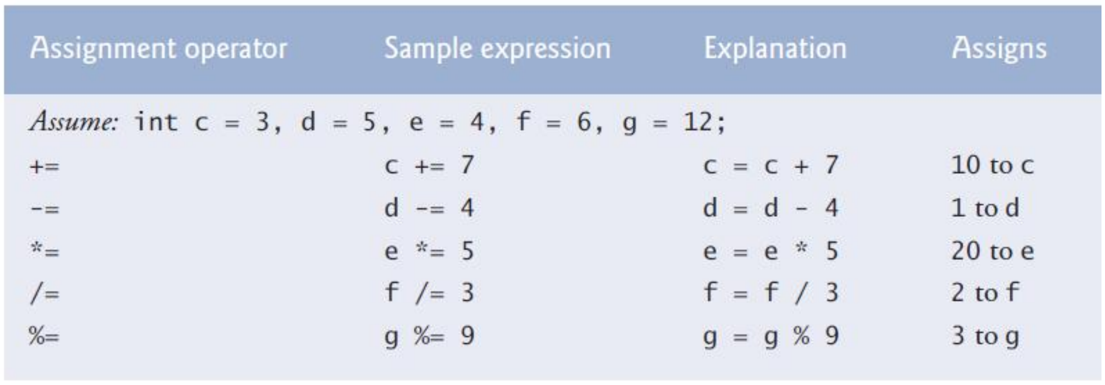
16. While repetition statement

1）while 和 for 语句执行 0 次或者多次在他们语句中的操作，如果循环条件最开始的时候是 false，语句不会执行（while and for statements perform the action(s) in their bodies zero or more times, if the loop-continuation condition is initially false, the body will not execute）

2）do…while 语句执行 1 次或者多次在他们语句中的操作（The do…while statement performs the action(s) in its body one or more times.）

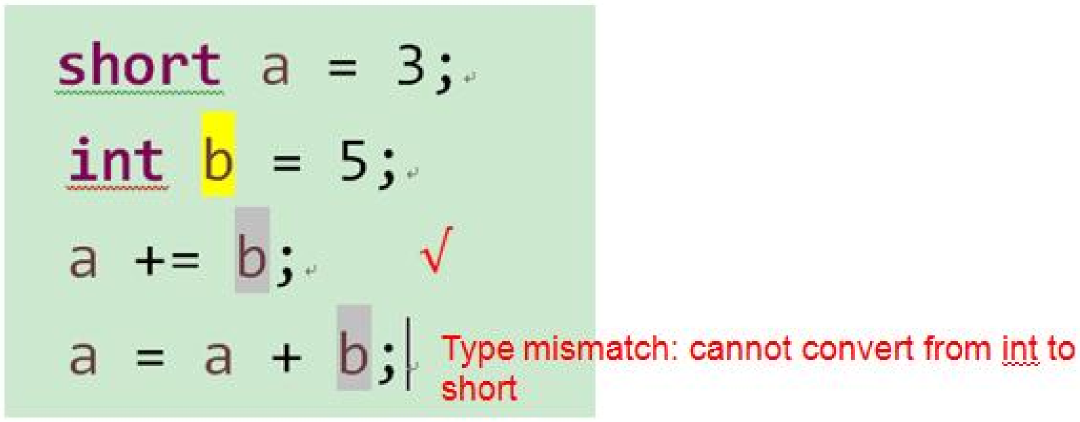
if, else, switch, while, do, for 都是关键字。

17. =　　+=　　-= …



Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Fig. 4.13 | Arithmetic compound assignment operators.

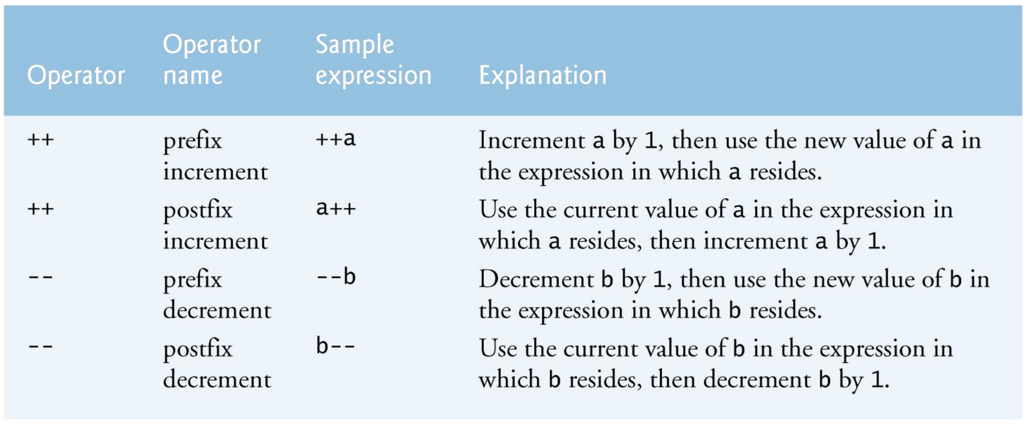


```
short a = 3;
int b = 5;
a += b;      √
a = a + b;
```

18.++　　--

一元增量运算符++，操作数加一（Unary increment operator, ++, adds one to its operand）

一元减量运算符--，操作数减一（Unary decrement operator, --, subtracts one from its operand）



Operator	Operator name	Sample expression	Explanation
++	prefix increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postfix increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	prefix decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postfix decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 4.14 | Increment and decrement operators.

19.Graphics, JPanel, paintComponet

类 Graphics 提供各种在屏幕上画文本或者图形的方法（Class Graphics (from package java.awt) provides various methods for drawing text and shapes onto the screen）

类 JPanel 提供一个可以画画的区域（Class JPanel (from package javax.swing) provides an area on which we can draw）

JPanel has a paintComponent method, which the system calls every time it needs to display the JPanel.

The first statement in every paintComponent method you create should always be   super.paintComponent(g);

常用绘图方法：

drawLine(int x1,int y1,int x2,int y2);

drawRect(int x,int y,int width,int height);

fillRect(int x,int y,int width,int height);//绘制填充矩形

drawRoundRect(int x,int y,int width, int height, int arcWidth, int arcHeight);//绘制圆角矩形

draw3DRect(int x,int y,int width,int height, boolean raised);//raised ==true,突出

drawArc(int x,int y,int width,int height,int startAngle, int arcAngle);//画弧，fill 画缺角的圆

clearRect(int x,int y, int width,int height);//变白

clipRect(int x,int y,int width,int height);//限制作图区域,多个则取交集

copyArea(int x,int y,int width,int height, int dx, int dy);//复制

drawPolygon(int xpoints[],int yPoints[],int nPoints);//多边形

20.for, switch, do…while

涉及到精确计算不能够用 double 和 float

For(int x:arrays)

X = A

Switch 语句中不能用 Double，string，对象等，可用 char，int，boolean

21.break and continue

break：跳出当前循环，从循环中提前退出

continue: 跳过循环体的剩余语句，继续进行下一次迭代

22.logical operators

& && | || ! ^

&& (逻辑与)

|| (逻辑或)

& (按位取与)

| (按位取或)

^ (按位异或)

! (逻辑反).

&与&&的区别：

(x>y)&&(y!=i++) 只计算前面

(x>y)&(y!=i++) 两边都计算

(x>y)&&(i++) 错误！整形不能提升为 boolean 型

23.static methods and fields

静态方法和实例方法的区别：

在外部调用静态方法时，可以使用"类名.方法名"的方式，也可以使用"对象名.方法名"的方式。而实例方法只有后面这种方式。也就是说，调用静态方法可以无需创建对象。

静态方法在访问本类的成员时，只允许访问静态成员（即静态成员变量和静态方法），而不允许访问实例成员变量和实例方法；实例方法则无此限制。

静态字段：静态字段表示在这块内存里定义了一个变量，不用 new，其他的地方一旦修改这个变量，在任何地方的这个变量的值都会被修改。

24.Argument Promotion and Casting

Double x=3 正确　　int x = 3.5 错误

Function(int x)<—3.5 错误　function(double x)<—3 正确

25.final

Final 放在 student 前面，那么 student 对象里面的变量还是可以修改的。

Final 和 static

1)final 类不可继承

2)static 和 private 的方法是 final 类型的，不可重写

3)构造函数不能定义为虚方法，因为也是 final 类型的

4)Final 定义的成员，必须要在定义的时候初始化，或者在构造函数中初始化。

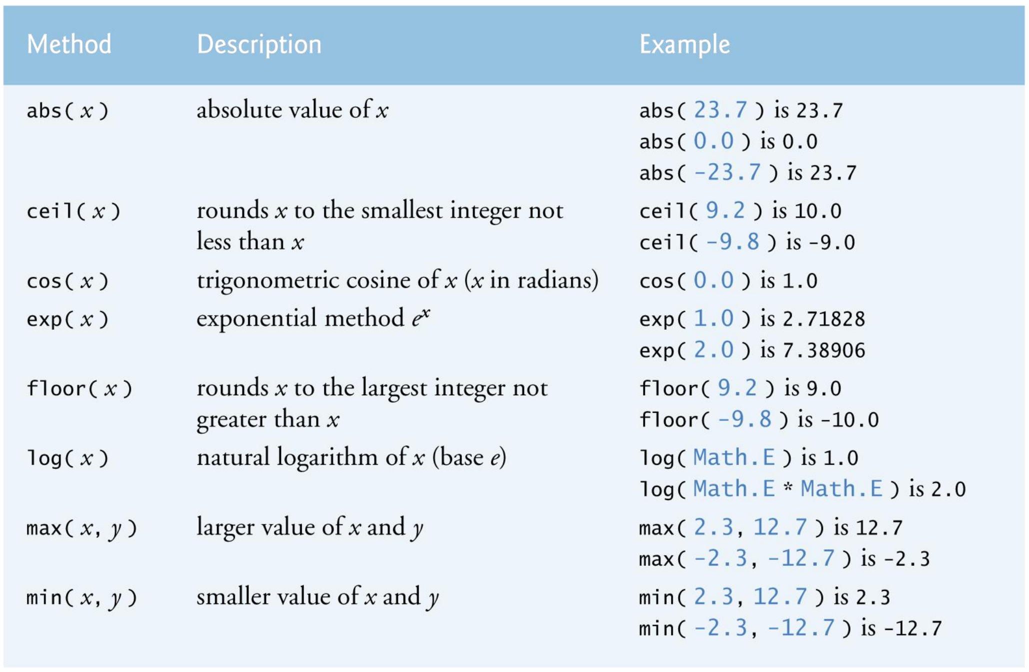
Final and finally and finalize 区别：

1)Final 用于声明属性,方法和类,分别表示属性不可变，方法不可覆盖，类不可继承

2)Finally 是异常处理语句结构的一部分，表示总是执行

3)Finalize 是 object 类的一个方法,在垃圾回收机制中会被自动调用于回收对象。

26.Class Math.



Method	Description	Example

`abs( x )`	absolute value of *x*	`abs( 23.7 )` is `23.7`
		`abs( 0.0 )` is `0.0`
		`abs( -23.7 )` is `23.7`

`ceil( x )`	rounds *x* to the smallest integer not less than *x*	`ceil( 9.2 )` is `10.0`
		`ceil( -9.8 )` is `-9.0`

`cos( x )`	trigonometric cosine of *x* (*x* in radians)	`cos( 0.0 )` is `1.0`

`exp( x )`	exponential method e^x	`exp( 1.0 )` is `2.71828`
		`exp( 2.0 )` is `7.38906`

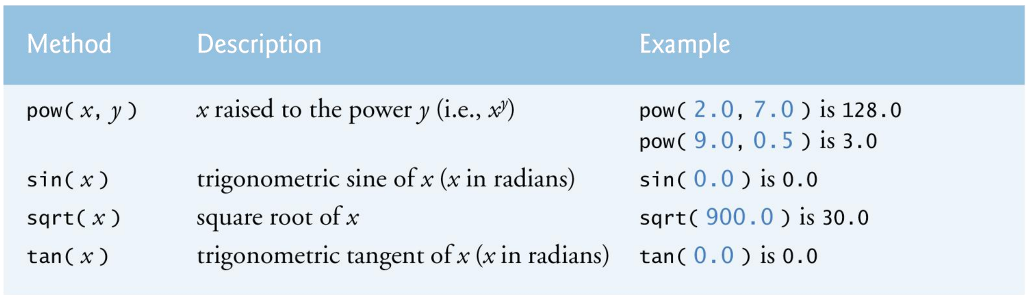
`floor( x )`	rounds *x* to the largest integer not greater than *x*	`floor( 9.2 )` is `9.0`
		`floor( -9.8 )` is `-10.0`

`log( x )`	natural logarithm of *x* (base *e*)	`log( Math.E )` is `1.0`
		`log( Math.E * Math.E )` is `2.0`

`max( x, y )`	larger value of *x* and *y*	`max( 2.3, 12.7 )` is `12.7`
		`max( -2.3, -12.7 )` is `-2.3`

`min( x, y )`	smaller value of *x* and *y*	`min( 2.3, 12.7 )` is `2.3`
		`min( -2.3, -12.7 )` is `-12.7`

Fig. 6.2 | `Math` class methods. (Part 1 of 2.)



Method	Description	Example

`pow( x, y )`	x raised to the power y (i.e., x^y)	`pow( 2.0, 7.0 )` is `128.0`
		`pow( 9.0, 0.5 )` is `3.0`

`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is `0.0`
`sqrt( x )`	square root of x	`sqrt( 900.0 )` is `30.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is `0.0`

Fig. 6.2 | Math class methods. (Part 2 of 2.)

Math.PI (3.141592653589793)

Math.E (2.718281828459045)

PI and E are declared final because their values never change.

27.passing information between methods.

基本数据类型和引用数据类型相同点都是值传递，不同是引用是传地址。

28.packages import

java.long 是默认包。包在前，inport 在后。

29.random-number generation

Math　Random

int x = 1+randomNumbers.nextInt(6);//生成一个[1, 7)之间的随机数

Math.random();是令系统随机选取大于等于 0.0 且小于 1.0 的伪随机 double 值

30.What method overloading is

一个类中两个或者多个方法同名，但参数列表不同

Overwriting and overloading:

方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现。重写 Overriding 是父类与子类之间多态性的一种表现，重载 Overloading 是一个类中多态性的一种表现。

如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。

如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。

Overloaded 的方法是可以改变返回值的类型。

31.To declare an array, initialize an array and refer to individual elements of an array.

Int[] c = new int[12];

Int[] c;

C = new int[12];

String b[] = new String[2]; //当声明只有一个变量的时候可以放变量后面，最好放前面

32.enhanced for statement

1）for (type 变量名：集合变量名 ）｛ … ｝

2）迭代变量必须在( )中定义！

3）集合变量可以是数组或实现了 Iterable 接口的集合类

33.To pass arrays to methods

34.multidimensional arrays

int a[][];//a[0]相当于一个 int[] ，也就是说，java 中的多维数组每一维的长度可以不一样。如：

int [][]a = {{1,2,3},{1},{2,1}};

int[] a,b,c　//a，b，c 都是 int[]

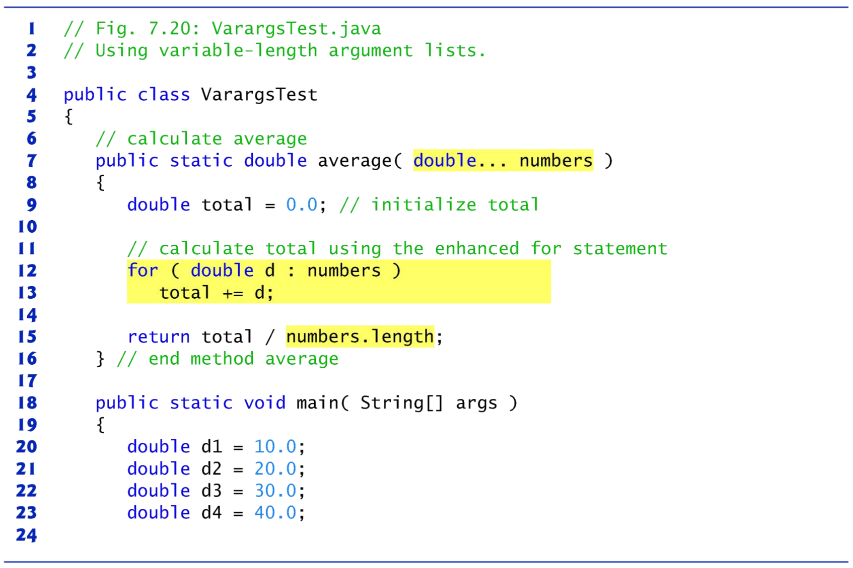
int a[], b, c　//a 是 int[],b 和 c 是 int

int x[];

x={1,2,3};错误

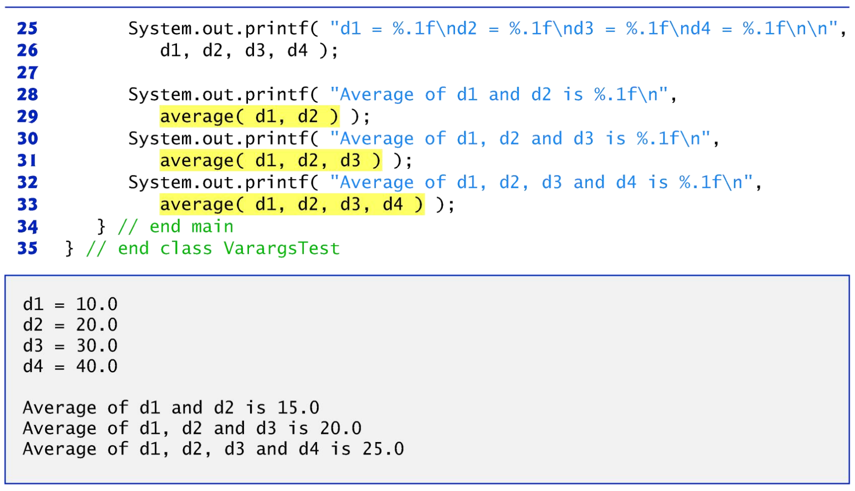
35.variable-length argument lists

可变长度的参数列表：…放在类型后面变量名前面，只能有一个。



```
// Fig. 7.20: VarargsTest.java
// Using variable-length argument lists.

public class VarargsTest
{
   // calculate average
   public static double average( double... numbers )
   {
      double total = 0.0; // initialize total

      // calculate total using the enhanced for statement
      for ( double d : numbers )
         total += d;

      return total / numbers.length;
   } // end method average

   public static void main( String[] args )
   {
      double d1 = 10.0;
      double d2 = 20.0;
      double d3 = 30.0;
      double d4 = 40.0;

```

Fig. 7.20 | Using variable-length argument lists. (Part 1 of 2.)



```java
      System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n",
         d1, d2, d3, d4 );

      System.out.printf( "Average of d1 and d2 is %.1f\n",
         average( d1, d2 ) );
      System.out.printf( "Average of d1, d2 and d3 is %.1f\n",
         average( d1, d2, d3 ) );
      System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
         average( d1, d2, d3, d4 ) );
   } // end main
} // end class VarargsTest
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

Fig. 7.20 | Using variable-length argument lists. (Part 2 of 2.)

36.Encapsulation and data hiding

37.this

Java 关键字 this 只能用于方法方法体内。当一个对象创建后，Java 虚拟机（JVM）就会给这个对象分配一个引用自身的指针，这个指针的名字就是 this。因此，this 只能在类中的非静态方法中使用，静态方法和静态的代码块中绝对不能出现 this。

this 放在方法前面：当做对象调用

类名.this;内部类调用外部类

38.Composition

has-a 关系：类可以引用其他类的对象，将它作为自己的成员组合比如一个 Date 类，然后另外 enployee 类引用了它的对象。

39.String.format

40.Garbage Collection

41.Static import

1）你必须写成 import static

2）提防含糊不清的命名 static 成员。例如，如果你对 Integer 类和 Long 类执行了静态导入，引用 MAX_VALUE 将会导致错误，因为 Integer 和 Long 类都有 MAX_VALUE 常量。

3）你可以在 static 对象引用，常亮（它们必须是 static 或者 final）和 static 方法上进行静态引入。

42.enum

enum 类型隐含为 final，是常量不能修改

enum 常量隐含为静态的

不能用 new 运算符创建 enum

```
Public enum Book
{
    JHTP(“java how to program”,”2012”);
    CHTP(“C how to program”,”2011”);
    Private final String title;
    Private finel String year;
    Book(String title,String year)
    {
        This.title = title;
        This.year = year;
    }
}
```

For(Book book:Book.values())

Cout<<book.getTitle()<<book.getYear();

43.The notions of superclasses and subclasses

创建类时，指定新类从现有类中继承某些成员，而不是完全地声明新的成员。现有类是超类，新类是子类。

44.super.

Super()调用父类构造函数和方法。

当一个子类方法覆盖父类继承方法,要访问子类的父类方法可以通过关键字 super 加上(.)(When a subclass method overrides an inherited superclass method, the superclass method can be accessed from the subclass by preceding the superclass method name with keyword super and a dot (.) separator)

45.How constructors are used in inheritance hierarchies

构造方法不会被继承

子类的构造方法在执行自己的任务之前，要显示地（super）或隐式地调用自己超类的构造方法。（The first task of a subclass constructor is to call its direct superclass’s constructor explicitly or implicitly）

如果代码没有显示的调用父类构造函数，就会隐式调用父类的默认或者无参数的构造函数（If the code does not include an explicit call to the superclass constructor, Java implicitly calls the superclass’s default or no-argument constructor）

一个类的默认构造函数是父类的默认或者无参数构造函数（A class’s default constructor calls the superclass’s default or no-argument constructor.）

46.Class Object

Clone 返回 object 引用，为它调用的对象创建一个副本

Equals 比较俩个对象的相等性

Finalize 内存回收器调用，执行终止清理

getClass 获取自己的类型

hashCode 哈希码是一个 int 值

wait，notify，notifyAll 多线程

toString 返回对象字符串表示

47.The concept of polymorphism and how to use it

多态概念：发送消息给某个对象，让对象自行决定影响何种行为。通过将子类对象引用赋值给超类对象引用变量来实现动态方法的调用。A a = new B();

多态作用：

应用程序不必为每一个子类编写功能调用，只需要对抽象基类进行处理即可。大大提高程序的可复用性。//继承

子类的功能可以被超类的方法或引用变量所调用，这叫向后兼容，可以提高可扩充性和可维护性。 //多态的真正作用，以前需要用 switch 实现

48.Overriding

final 方法不能被重写，声明静态的方法是隐式的 final

49.abstract class, interface and concrete classes

接口和抽象类区别：

1）抽象类的方法不一定是抽象的

2）接口只含有常量和抽象方法，而且均为 public

3）抽象类的子类需要重写虚方法

4）接口不需要重写虚方法。

5）一组相关类，抽取相同属性组成接口？错误 是抽象类

接口和抽象类之间的关系：

相同点：

（1)接口和抽象类都不能被实例化,它们都位于继承树的顶端，用于被其他的类实现和继承。

（2）接口和抽象类都是可以包含抽象方法的，实现接口或是继承抽象类的普通子类都必须实现这些抽象方法。

不同点：

（1）接口只能包含抽象方法，不能包含已经提供实现的方法；抽象类则完全可以包含普通的方法

（2）接口不能定义静态方法；抽象类完全可以定义静态方法。

（3）接口中只能定义静态常量 Field，不能定义普通的 Field；抽象类既可以定义普通的 Field 也能定义静态常量 Field

（4）接口不能包含构造器；抽象类中可以包含构造器，抽象类中的构造器并不是用于创建对象的，而是让其子类调用这些构造器来完成抽象类的初始化操作。

（5）接口里面不能够包含初始化块；但是抽象类里面则完全可以包含初始化块

（6）一个类最多只能有一个直接父类，包括抽象类；但是一个类可以直接实现多个接口，通过实现多个接口可以弥补 Java 中的单继承的不足。

50.To determine an object's type at execution time

instanceof

```
If（A instanceof B）
{
    //如果 对象 A 是 类 B 的实例，则
}
```

注意： 还有，A 是 B 的子类的对象也可以

51.try, throw, catch and finally

1）try 语句块后面必须至少紧跟着一个 catch 语句块或一个 finally 语句块。

2）每个 catch 语句块只有一个参数。

3）存在多个 catch 捕获时，只执行第一个匹配的 catch 语句块。

4）不可以在子类的 catch 语句块之前放置超类的语句块。

Throw 和 throws：

1）throw 是语句抛出一个异常。

throws 是方法可能抛出异常的声明，抛出多个异常。(用在声明方法时，表示该方法可能要抛出异常)

2）throws 出现在方法函数头；而 throw 出现在函数体。

throws 表示出现异常的一种可能性，并不一定会发生这些异常；throw 则是抛出了异常，执行 throw 则一定抛出了某种异常。

3） throw 语句用在方法体内，表示抛出异常，由方法体内的语句处理。

throws 语句用在方法声明后面，表示再抛出异常，由该方法的调用者来处理

Finally：

1）Try 语句是否抛出异常都会执行，try 使用了 return、break、continue 都会执行

但是使用 System.exit()方法退出时，不会执行 finally

2）finally 语句块是释放在 try 语句块中分配的资源的理想场所，这样有助于消除资源的泄漏

3）避免在 finally 和 catch 放置能够抛出异常的代码。

52.Checked exceptions and unchecked exceptions

1）除了 RuntimeException，其他继承自 java.lang.Exception 得异常统称为 Checked Exception。简单的说：Checked 就是在 compile 阶段会检查，必须用 try catch 或者 throws 处理，比如 FileNotFoundException。

2）Unchecked 就是 compile 阶段不会检查，即 RuntimeException（运行时异常）比如 ArithmeticException 除法（除以 0）。

53.JFrame, JLabel, JTextField, JPasswordField, JButton, JPanel

JFrame：容器，一般不直接将组件放在里面

JLabel：文本标签

JTextField：文本框

JPasswordField：密码框

JButton：按钮

JPanel：面板，可直接在上面放组件

54.JCheckBox, JRadioButton, JComboBox, JList

JCheckBox：多选按钮

JRadioButton：单选按钮——ButtonGroup

JComboBox：列表选择框，参数是字符串数组

JList：项目序列，可以选择一个或多个项目

55.event-handling classes and interfaces

在接口中定义变量必须初始化，因为它默认的是 public, static, final。方法默认是 public, abstract

事件处理的主要步骤：

1）首先产生监听器，监听是否有事件源的产生。

2）在要实现具有事件处理功能的组件上注册监听器。

3）注册监听器的组件产生事件源，并将事件对象返回给监听器。

4）有监听器调用相应的方法处理事件。

a.自身类作为监听器（通过接口实现）

```
class ThisClassEvent extends JFrame implements ActionListener{
    btn.addActionListener(this);
    public void actionPerformed (ActionEvent e){
        Container c=getContentPane();
        c.setBackground(Color.red);
    }
}
```

b.外部类作为事件监听器

```
btn.addActionListener(new OuterClass(this));
class OuterClass implements ActionListener{
    OuterClassEvent oce;
    public OuterClass(OuterClassEvent oce){
        this.oce = oce;
    }
    public void actionPerformed(ActionEvent e){
        Container c=oce.getContentPane();
        c.setBackground(Color.red);
    }
}
```

c.匿名内部类作为事件监听器:

```
btn.addActionListener(new ActionListener(){
```

```
          public void actionPerformed(ActionEvent e){
                    Container c=getContentPane();
                    c.setBackground(Color.red);
          }
        });
```

d.内部类作为事件监听器:

```
btn.addActionListener(new InnerClass());
class InnerClass implements ActionListener{
        public void actionPerformed (ActionEvent e){
                Container c=getContentPane();
                c.setBackground(Color.red);
        }
}
```

56.BorderLayout, FlowLayout

BorderLayout()建立一个具有东西南北中五个方位的布局

FlowLayout()建立一个默认为居中对齐,组件彼此有 5 单位的水平与垂直间距的 FlowLayout

FlowLayout(int align, int hgap, int vgap)

建立一个可设置排列方式与组件间距的 FlowLayout

三个参数：对其方式，水平间隙，垂直间隙

57.Draw line…

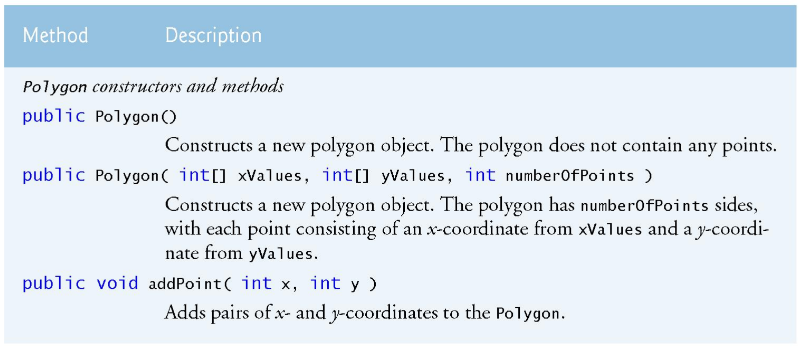
58.Polygons and Polylines

Method	Description
`Graphics` methods for drawing polygons	
`public void drawPolygon( int[] xPoints, int[] yPoints, int points )`	Draws a polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.
`public void drawPolyline( int[] xPoints, int[] yPoints, int points )`	Draws a sequence of connected lines. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. If the last point is different from the first, the polyline is not closed.
`public void drawPolygon( Polygon p )`	Draws the specified polygon.

Fig. 15.26 | `Graphics` methods for polygons and class `Polygon` methods. (Part 1 of 3.)

Method	Description
`public void fillPolygon( int[] xPoints, int[] yPoints, int points )`	Draws a filled polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.
`public void fillPolygon( Polygon p )`	Draws the specified filled polygon. The polygon is closed.

Fig. 15.26 | `Graphics` methods for polygons and class `Polygon` methods. (Part 2 of 3.)



Method	Description
`Polygon` constructors and methods	
`public Polygon()`	Constructs a new polygon object. The polygon does not contain any points.
`public Polygon( int[] xValues, int[] yValues, int numberOfPoints )`	Constructs a new polygon object. The polygon has `numberOfPoints` sides, with each point consisting of an *x*-coordinate from `xValues` and a *y*-coordinate from `yValues`.
`public void addPoint( int x, int y )`	Adds pairs of *x*- and *y*-coordinates to the `Polygon`.

Fig. 15.26 | `Graphics` methods for polygons and class `Polygon` methods. (Part 3 of 3.)

59.Color, JColorChooser, Font

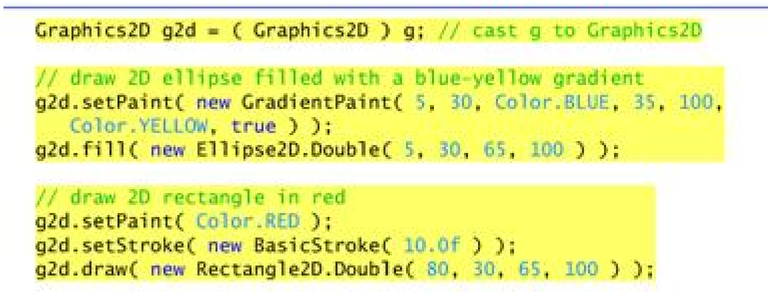
JColorChooser： color = JColorChooser.showDialog(component,”name” ,默认 color);

Font：font = new Font(“字体” , 粗细, 大小);

Color： color = new Color(int r,int g,int b);

60.Graphics2D

draw.



```
Graphics2D g2d = ( Graphics2D ) g; // cast g to Graphics2D

// draw 2D ellipse filled with a blue-yellow gradient
g2d.setPaint( new GradientPaint( 5, 30, Color.BLUE, 35, 100,
   Color.YELLOW, true ) );
g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );

// draw 2D rectangle in red
g2d.setPaint( Color.RED );
g2d.setStroke( new BasicStroke( 10.0f ) );
g2d.draw( new Rectangle2D.Double( 80, 30, 65, 100 ) );
```

61.Paint ,Stroke

1)Ged.setPaint(new GradientPaint(5,30,Color.BLUE,35,100,Color.YELLOW,true));

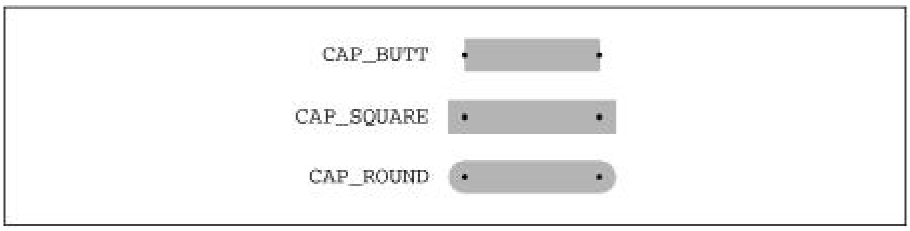
GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2) //构造一个简单的非周期性 GradientPaint 对象。

GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic) //根据 boolean 参数构造一个周期性或非周期性的 GradientPaint 对象。

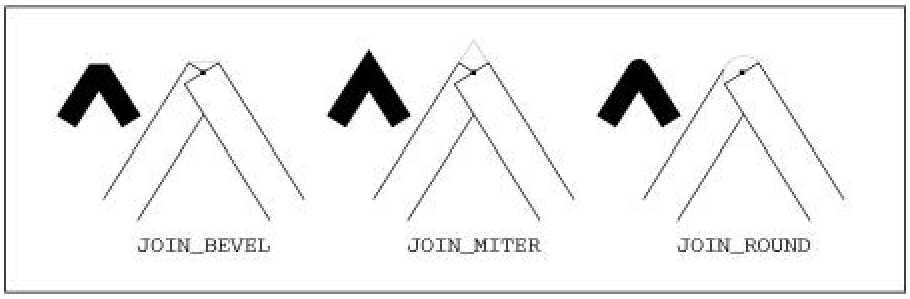
2)G2d.setStroke(new BasicStroke(4,BasicStroke.CAP_ROUND,BasicStroke.JOIN_ROUND,10,dashes,0));

```
public BasicStroke(float width, int cap, int join, float miterlimit,
            float dash[], float dash_phase) {
```

cap 只能取三个值：CAP_BUTT, CAP_ROUND or CAP_SQUARE，表示不同的线端。



join 这个参数表示当两条线连接时，连接处的形状，可以取 JOIN_ROUND, JOIN_BEVEL, or JOIN_MITER 三个值。

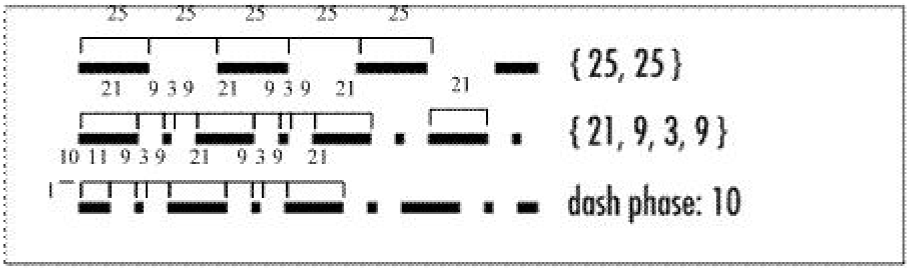


miterlimit 当你使用 JOIN_MITER 这种策略来表示连接处形状的时候，由于两根线的连接角度可能很小，那么，就会导致延伸出来的那个角特别长！本参数用来限制那个尖角的最大长度！

当你使用 JOIN_MITER 策略的时候，本参数必须大于 1，本参数的默认值是 10.0f 。

dash_phase 最后一个参数是跟 dash[]这个数组配合的参数，表示在画虚线的时候，从一定的偏移量处开始画。

下面这个图，统一说明了 dash[]和 dash_phase 的意思：



62.String

String s1 = new String(charArray,6,3)//截取从 6 开始的 3 个字符

String 方法：

1）Length() 返回字符串长度

2）charAt(int) 获得字符串指定位置的字符

3）getChars 取得作为 char 数组返回的字符串中的字符集

4）equals(string) 比较字符串中每一个 Unicode，区分大小写

5）== 比较引用是否指向同一个对象

6）equalsIgnoreCase(string) 不区分大小写比较

7）compareTo(string) 相等返回 0，A 大于 B 返回正，A 小于 B 返回-1；

8）regionMatches(boolean,A 起始索引,string,B 起始索引,比较个数); true 时忽略大小写。

9）startWith(string,int) 比较改字符串是否 int 位置是 string。Int 可省略为从头开始

10）endWith(string) 比较字符串是否是以 sting 结尾。

其他方法：

1）indexOf(‘a’,int x); 搜索字符串首次出现’a’的值,x 索引位置

2）substring(20) 取从 20 索引开始到末尾的字符

3）substring(3,6) 去从 3 索引开始但不包括 6 的字符

4）concat(string) 拼接字符串

5）replace(‘1’,’L’) 用 L 换所有的 l

6）toUpperCase() 转换成大写

7）strim() 消除字符串的开始和末尾的空白符

8）valueOf(charArray,int x,int y) 返回从 x 开始 y 个字符进入 charArray 数组中

63.StringBuilder

用于创建和操作动态字符串信息——可修改的字符串

为什么要用 StringBuilder?

1）String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象, 然后将指针指向新的 String 对象, 所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当

内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。

2）而如果是使用 StringBuilder 类则结果就不一样了，每次结果都会对 StringBuilder 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuilder，特别是字符串对象经常改变的情况下。

主要方法：

1）append() 实现+,+=操作的字符串拼接

2）insert(int x,string) 指定 x 索引出插入 string

3）deleteCharAt(int x) 删除指定 x 索引的字符

64.Character

An integer value represented as a character in single quotes.

The value of a character literal is the integer value of the character in the Unicode character set.

65.Regular Expression

String.matches(“正则表达式”);

Pattern express = Pattern.compile(“正则表达式”);

Matcher matcher = express.matches(string);

While(matcher.find())

…..

\d 任何数字　\D 任何非数字　\w 任何词字符　\W 任何非词字符

\s 任何空白符　　\S 任何非空符　^s 除了 s 之外的所有字符

* 匹配前面模式 0-n 次　　＋ 匹配前面模式 1-n 次

? 匹配前面模式 0-1 次　　{n}正好匹配 n 次

{n,} 至少匹配 n 次　　{n,m}匹配 n-m 次

匹配示例：

[A-Z]：匹配单个大写字符　[A-Za-z]：匹配大小写字符

[^Z]与[A-Y]不一样，前者还匹配小写字符和非字母

[A-Z][a-zA-Z]* 名字 Jane

[\\d+\\s+([a-zA-Z]+ | [a-zA-Z]+ \\s[a-zA-Z]+)] 123 Some Street

[\\d{5}] 12345

66.To create, read, write and update files

Create：

File myFile = new File(string)——四种方式（1.一个字符串指定名称和路径 2.第一个字符串指定路径第二个字符串指定名称 3.对象 4.URL）

Read and write:

顺序存取就是用 Scanner 和 Formatter 类按照一定的顺序读取结构化文件。（java 视文件是无结构的）

1）写:

```
Formatter   ouput = new Formatter("clients.txt");
output.format("%d %s %s %.2f\n",string1,string2,string3,double1);
```

2）读：

Scanner   input = new Scanner(new File(“clients.txt”));

String1 = input.next();

Int1 = input.netInt();

对象序列化：

1）写：

FileOutputStream output = new FileOutputStream(“clients.txt”);

output.writeObject(string);

注意:用 FileOutputStream 打开已经存在的文件输出，则它以前的内容会被清除

2）读：

FileInputStream input = new FileInputStream(“clients.txt”);

System.out.println(input.readObject.getName());

调用 readObject 方法从文件读取对象进入自己定义的继承 Serializable 接口的对象。

注意：用 FileInputStream 从文件读取数据时，必须与写入他们时是相同的格式.

67.The differences between text files and binary files

1）基于字节的流以二进制格式输入和输出数据，而基于字符的流以字符序列输入和输出数据。

2）值 5 为例，基于字节的流存储——>101

基于字符的流存储——>0000000000110101（53 的二进制）

3）用基于字节的流创建的文件，称为二进制文件（binary files）

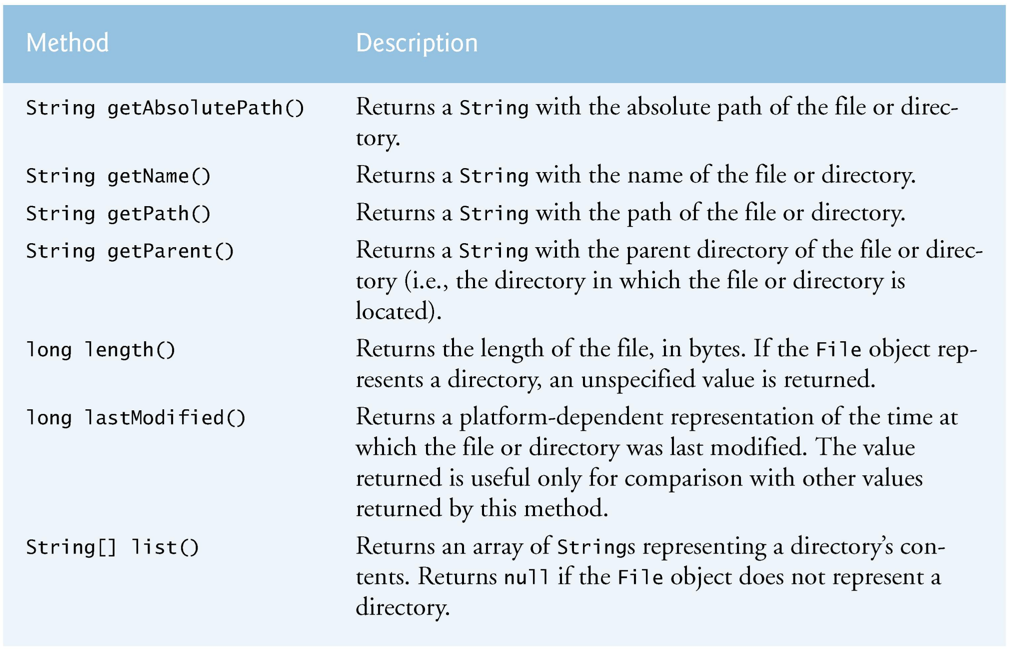
用基于字符的流创建的文件，称为文本文件（textfiles）

4）文本文件可以由文本编辑器读取，而二进制文件是由能够理解文件的特点内容以及它的顺序的程序读取的。

68.File

Method	Description
`boolean canRead()`	Returns `true` if a file is readable by the current application; `false` otherwise.
`boolean canWrite()`	Returns `true` if a file is writable by the current application; `false` otherwise.
`boolean exists()`	Returns `true` if the file or directory represented by the `File` object exists; `false` otherwise.
`boolean isFile()`	Returns `true` if the name specified as the argument to the `File` constructor is a file; `false` otherwise.
`boolean isDirectory()`	Returns `true` if the name specified as the argument to the `File` constructor is a directory; `false` otherwise.
`boolean isAbsolute()`	Returns `true` if the arguments specified to the `File` constructor indicate an absolute path to a file or directory; `false` otherwise.

Fig. 17.2 | `File` methods. (Part 1 of 2.)



Method	Description
`String getAbsolutePath()`	Returns a `String` with the absolute path of the file or directory.
`String getName()`	Returns a `String` with the name of the file or directory.
`String getPath()`	Returns a `String` with the path of the file or directory.
`String getParent()`	Returns a `String` with the parent directory of the file or directory (i.e., the directory in which the file or directory is located).
`long length()`	Returns the length of the file, in bytes. If the `File` object represents a directory, an unspecified value is returned.
`long lastModified()`	Returns a platform-dependent representation of the time at which the file or directory was last modified. The value returned is useful only for comparison with other values returned by this method.
`String[] list()`	Returns an array of `String`s representing a directory's contents. Returns `null` if the `File` object does not represent a directory.

Fig. 17.2 | `File` methods. (Part 2 of 2.)

69.Scanner , Formatter

Formatter 对象会输出格式字符串，具有与 System.out.printf 方面相同的格式化能力。但是 Formatter 对象可以输出到各种地方，比如屏幕和文件。

70.FileInputStream , FileOutputStream

71.JFileChooser , ObjectInputStream, ObjectOutputStream

ObjectInputStream 和 ObjectOutputStream 类分别实现了 ObjectInput 和 ObjectOutput 接口，使得他们能够从流中读取或写入全部对象。

JFileChooser:

JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

int result = fileChooser.showOpenDialog(this);//显示选择文件对话框

if(result == JFileChooser.CANCEL_OPTION)//判断是否选择好了文件

    System.exit(1);

    File filename = fileChooser.getSelectedFile();//获取文件对象

重点流类名及其作用：

1）FileInputStream 用于基于字节的输入

2）FileOutPutStream 用于基于字节的输出

3）BufferedOutputSteam 基于字节的缓冲输出

4）BufferedIntputSteam 基于字节的缓冲输入

5）PrintStream 基于字节的打印输出

6）PrintWriter 基于字符的打印输出

7）BufferedReader 基于字符的输入

8）BufferedWriter 基于字符的输出

9）FileReader 用于基于字符的输入

10）FileWriter 用于基于字符的输出

11）SequeInputStream 一次性读入多个文件

12）ZipCompress 生产压缩文件

文件打开错误异常：

1）SecurityException 用户没有向文件写入数据的权限

2）FileNotFoundException 文件不存在

文件的随机存取：

RandomAccessFile 类

RandomAccessFile(String filename,String mode) //创建从中读取和向其中写入（可选）的随机存取文件流，该文件具有指定名称。

文件存取通常是循序的，每在文件中存取一次，文件的读取位置就会相对于目前的位置前进一次。然而有时必须指定文件的某个区段进行读取或写入的动作，也就是进行随机存取(Random Access)，即要能在文件中随意地移动读取位置。这时可以使用 RandomAccessFile，使用它的 seek()方法来指定

72.What collections are

集合是一种数据结构（实际上就是对象），它保存其他对象的引用。通常，集合包含的引用，其对象都具有相同的类型。

73.Arrays, Collections

1）数组转 collection：

String [] str={"aa", "bb","cc"};

List list=Arrays.asList(str);

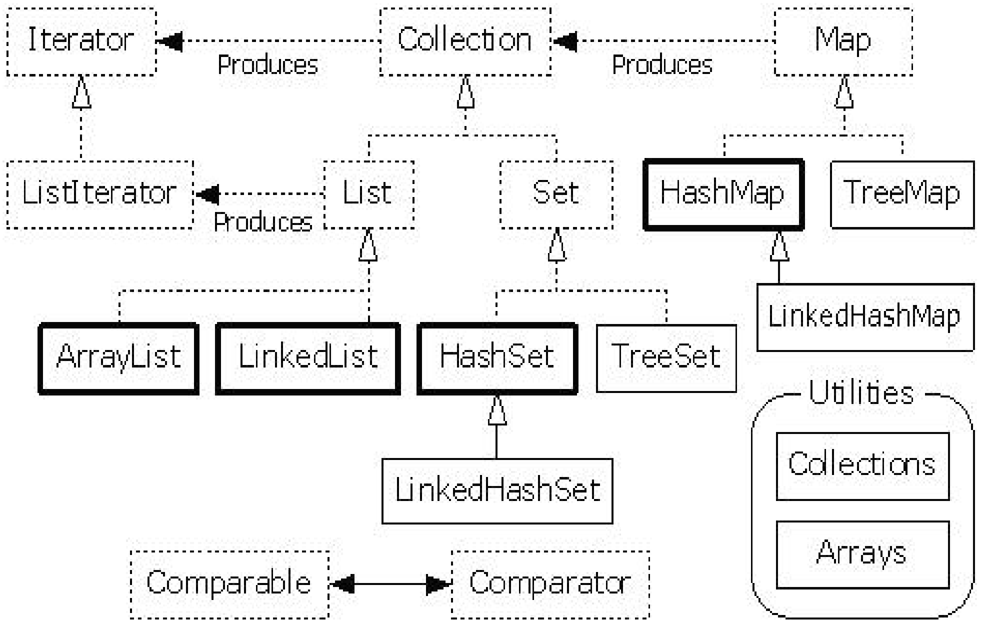
通过调用工具类 Arrays 的 asList 静态方法将数组转为 list ，这时 list 的长度为参数数组的大小，不能通过 add 来添加元素。当然也可以通过 for 循环来逐个向 list 里添加元素。

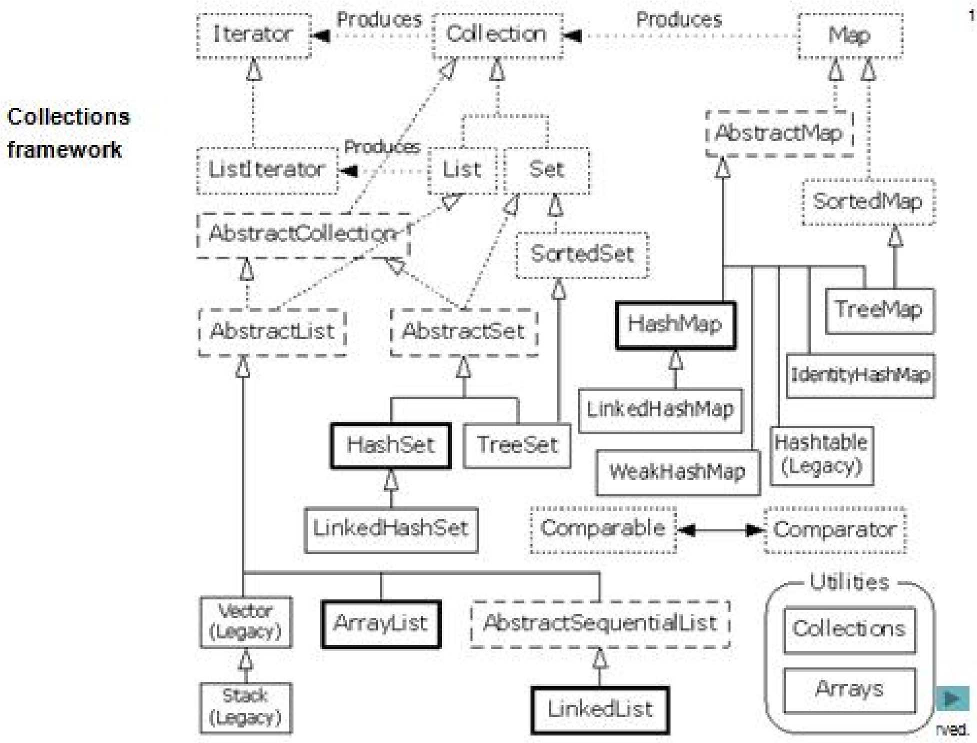
2）Collection 转数组

toArray()   // 返回 Object[]

toArray(T[] array)　//返回 T[]

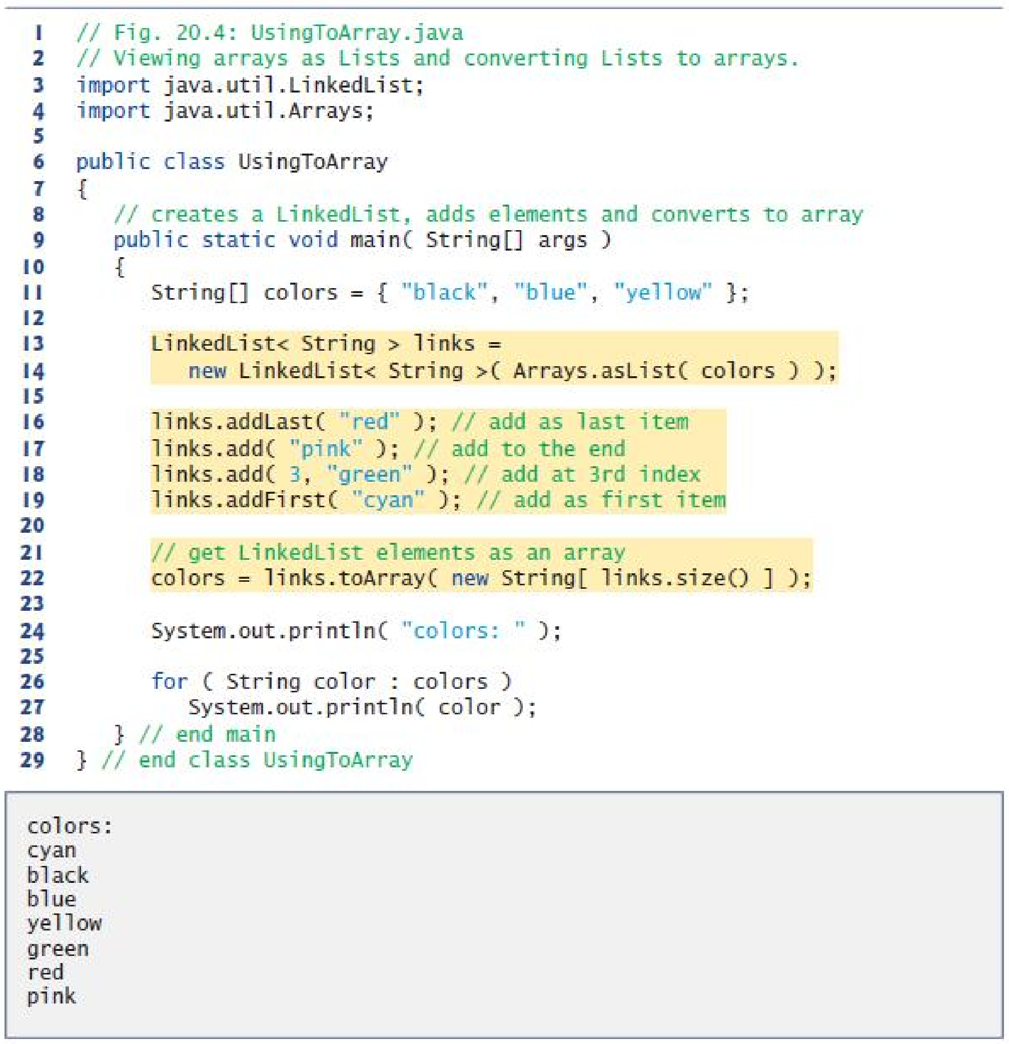
74.collections framework ,the difference between them





集合框架的一个重要特性：不管集合内部表示如何，都能够通过不同的集合类型操作某一个集合类型中的元素。通过一套公共方法操作的集合，称为视图。

Arrays.asList(数组对象)　//此静态方法用于将 Array 转化为 List 类型对象。常常用于 List 类型对象的初始化中。



```
// Fig. 20.4: UsingToArray.java
// Viewing arrays as Lists and converting Lists to arrays.
import java.util.LinkedList;
import java.util.Arrays;

public class UsingToArray
{
   // creates a LinkedList, adds elements and converts to array
   public static void main( String[] args )
   {
      String[] colors = { "black", "blue", "yellow" };

      LinkedList< String > links =
         new LinkedList< String >( Arrays.asList( colors ) );

      links.addLast( "red" ); // add as last item
      links.add( "pink" ); // add to the end
      links.add( 3, "green" ); // add at 3rd index
      links.addFirst( "cyan" ); // add as first item

      // get LinkedList elements as an array
      colors = links.toArray( new String[ links.size() ] );

      System.out.println( "colors: " );

      for ( String color : colors )
         System.out.println( color );
   } // end main
} // end class UsingToArray
```

```
colors:
cyan
black
blue
yellow
green
red
pink
```

注意：将包含数据的数组传递给 toArray 方面时，会导致逻辑错误。

1)如果数组中元素个数少于调用 toArray 方法的列表元素个数，则会分配一个新数组来保存列表中的元素。

2)如果数组中元素个数大于调用 toArray 方法的列表元素个数，则数组中元素会被列表中的元素覆盖掉。

75.What is functional interfaces

Java SE8 中任何只包含一个抽象方法的接口被称为函数式接口（As of Java SE 8, any interface containing only one abstract method is known as a functional interface）

76.Lambda Syntax

(parameterList) -> {statements}

举例：

(int x, int y) -> {return x + y;}

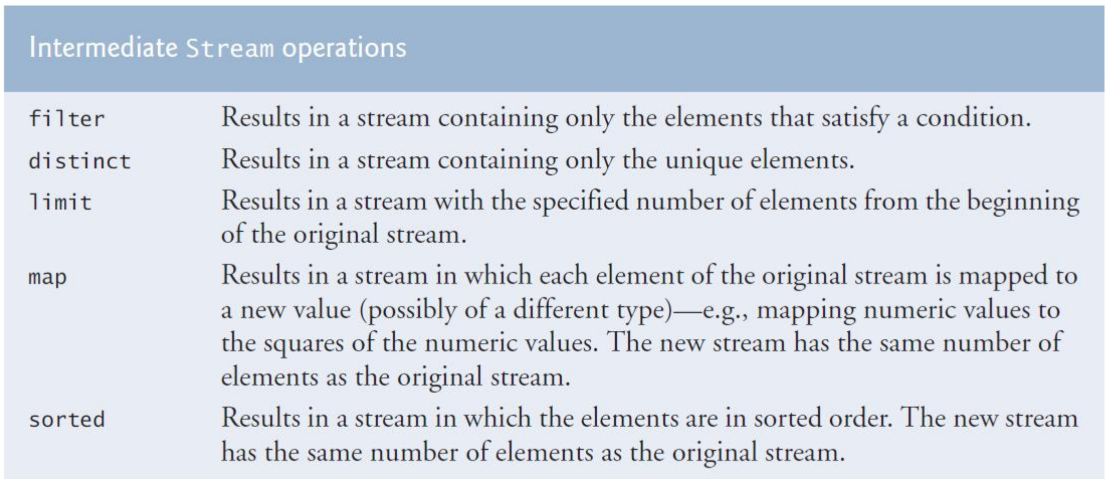
(x, y) -> {return x + y;}

(x, y) -> x + y

value -> System.out.printf("%d ", value)

() -> System.out.println("Welcome to lambdas!")

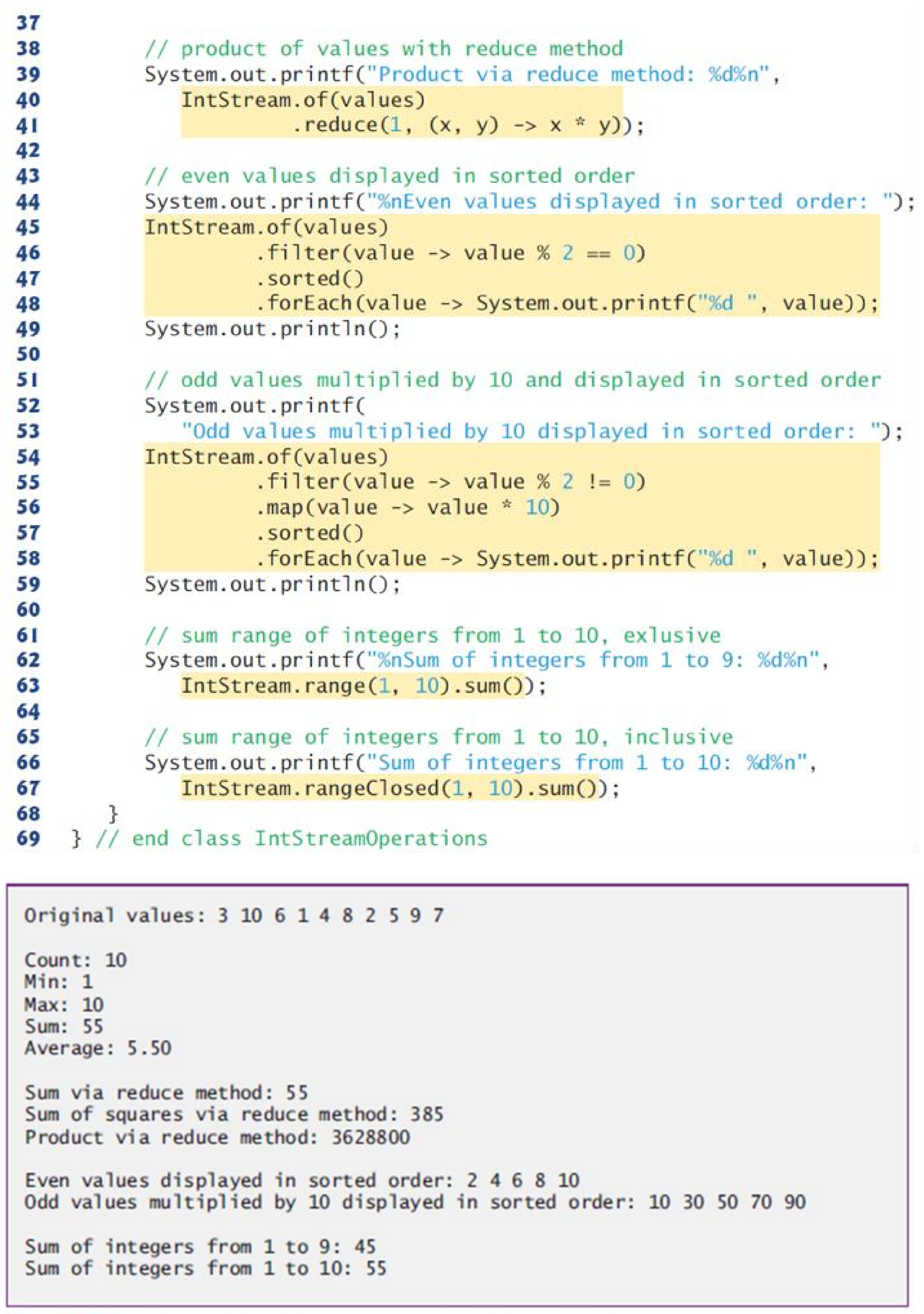
77.Methods of Streams



Intermediate `Stream` operations	
`filter`	Results in a stream containing only the elements that satisfy a condition.
`distinct`	Results in a stream containing only the unique elements.
`limit`	Results in a stream with the specified number of elements from the beginning of the original stream.
`map`	Results in a stream in which each element of the original stream is mapped to a new value (possibly of a different type)—e.g., mapping numeric values to the squares of the numeric values. The new stream has the same number of elements as the original stream.
`sorted`	Results in a stream in which the elements are in sorted order. The new stream has the same number of elements as the original stream.

Terminal `Stream` operations	
`forEach`	Performs processing on every element in a stream (e.g., display each element).
Reduction operations—Take all values in the stream and return a single value	
`average`	Calculates the *average* of the elements in a numeric stream.
`count`	Returns the *number of elements* in the stream.
`max`	Locates the *largest* value in a numeric stream.
`min`	Locates the *smallest* value in a numeric stream.
`reduce`	Reduces the elements of a collection to a *single value* using an associative accumulation function (e.g., a lambda that adds two elements).
Mutable reduction operations—Create a container (such as a collection or `StringBuilder`)	
`collect`	Creates a *new collection* of elements containing the results of the stream's prior operations.
`toArray`	Creates an *array* containing the results of the stream's prior operations.
Search operations	
`findFirst`	Finds the *first* stream element based on the prior intermediate operations; immediately terminates processing of the stream pipeline once such an element is found.
`findAny`	Finds *any* stream element based on the prior intermediate operations; immediately terminates processing of the stream pipeline once such an element is found.
`anyMatch`	Determines whether *any* stream elements match a specified condition; immediately terminates processing of the stream pipeline if an element matches.
`allMatch`	Determines whether *all* of the elements in the stream match a specified condition.

```java
// Fig. 17.5: IntStreamOperations.java
// Demonstrating IntStream operations.
import java.util.Arrays;
import java.util.stream.IntStream;

public class IntStreamOperations
{
   public static void main(String[] args)
   {
      int[] values = {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};

      // display original values
      System.out.print("Original values: ");
      IntStream.of(values)
               .forEach(value -> System.out.printf("%d ", value));
      System.out.println();

      // count, min, max, sum and average of the values
      System.out.printf("%nCount: %d%n", IntStream.of(values).count());
      System.out.printf("Min: %d%n",
         IntStream.of(values).min().getAsInt());
      System.out.printf("Max: %d%n",
         IntStream.of(values).max().getAsInt());
      System.out.printf("Sum: %d%n", IntStream.of(values).sum());
      System.out.printf("Average: %.2f%n",
         IntStream.of(values).average().getAsDouble());

      // sum of values with reduce method
      System.out.printf("%nSum via reduce method: %d%n",
         IntStream.of(values)
                  .reduce(0, (x, y) -> x + y));

      // sum of squares of values with reduce method
      System.out.printf("Sum of squares via reduce method: %d%n",
         IntStream.of(values)
                  .reduce(0, (x, y) -> x + y * y));
```



```java

        // product of values with reduce method
        System.out.printf("Product via reduce method: %d%n",
           IntStream.of(values)
                    .reduce(1, (x, y) -> x * y));

        // even values displayed in sorted order
        System.out.printf("%nEven values displayed in sorted order: ");
        IntStream.of(values)
                 .filter(value -> value % 2 == 0)
                 .sorted()
                 .forEach(value -> System.out.printf("%d ", value));
        System.out.println();

        // odd values multiplied by 10 and displayed in sorted order
        System.out.printf(
           "Odd values multiplied by 10 displayed in sorted order: ");
        IntStream.of(values)
                 .filter(value -> value % 2 != 0)
                 .map(value -> value * 10)
                 .sorted()
                 .forEach(value -> System.out.printf("%d ", value));
        System.out.println();

        // sum range of integers from 1 to 10, exlusive
        System.out.printf("%nSum of integers from 1 to 9: %d%n",
           IntStream.range(1, 10).sum());

        // sum range of integers from 1 to 10, inclusive
        System.out.printf("Sum of integers from 1 to 10: %d%n",
           IntStream.rangeClosed(1, 10).sum());
    }
} // end class IntStreamOperations
```

```
Original values: 3 10 6 1 4 8 2 5 9 7

Count: 10
Min: 1
Max: 10
Sum: 55
Average: 5.50

Sum via reduce method: 55
Sum of squares via reduce method: 385
Product via reduce method: 3628800

Even values displayed in sorted order: 2 4 6 8 10
Odd values multiplied by 10 displayed in sorted order: 10 30 50 70 90

Sum of integers from 1 to 9: 45
Sum of integers from 1 to 10: 55
```

Fig. 17.5 | Demonstrating `IntStream` operations. (Part 2 of 2.)

78.Generic class, generic methods

泛型方法使用户可以在一个方法声明中指定一组相关的方法

泛型类使用户可以在一个类声明中指定一组相关的类。

泛型还提供了编译时类型安全，能够在编译时捕获到无效的类型。

```
Private static<T> void printArray(T[] inputArray);
Private static<T extends Comparable<T>> T max(T x,T y,T z);
Public class Stack<T>
{
    Private ArrayList<T> elements;
}
```

79.To understand raw types（原始类型） and how they help achieve backwards compatibility

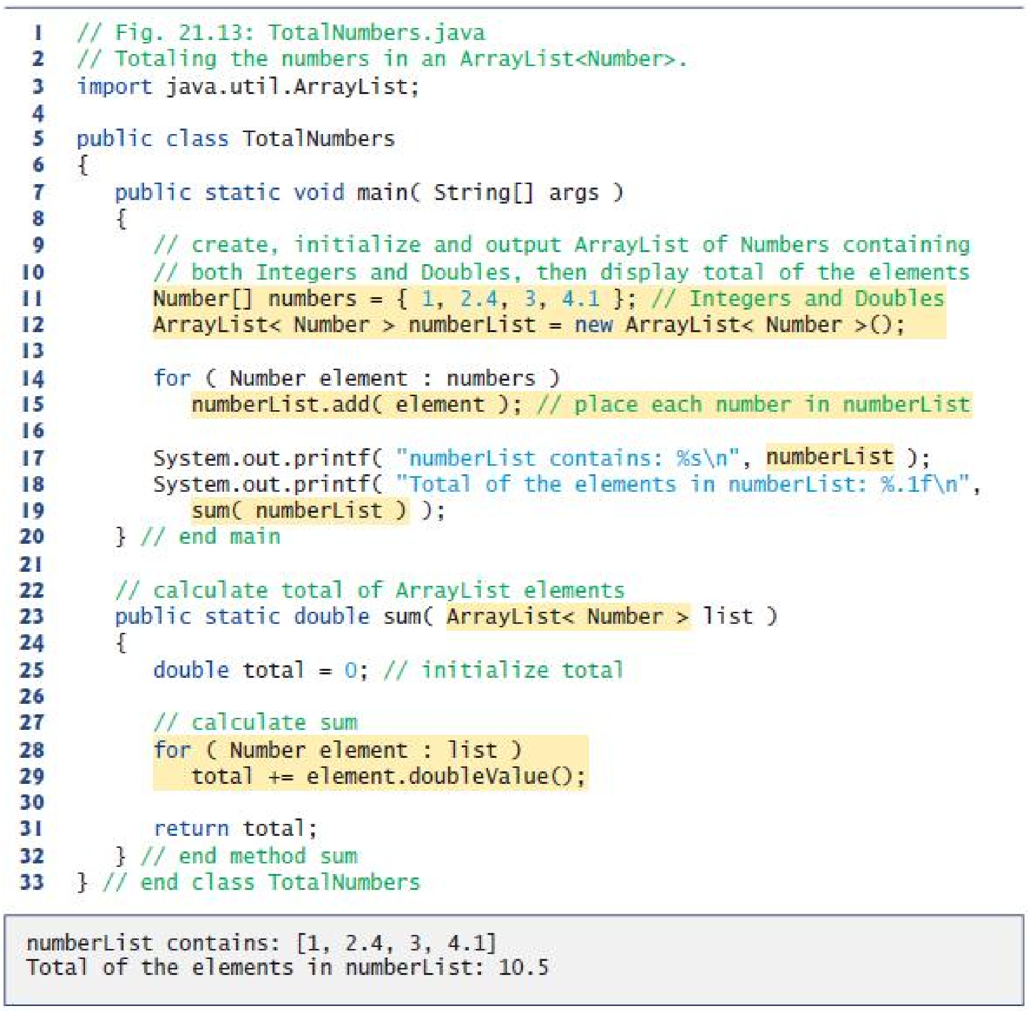
形如：

Stack objectStack = new Stack（5）；能够保存任何类型 object 类型

Stack objectStack = new <Double>Stack（5）; //被赋予指定了类型实参的 Stack

Stack <Double> objectStack = newStack（5）;//声明中指定了类型实参的 Stack，被赋予一个原始的 Stack 变量

80.To use wildcards(通配符) when precise type information about a parameter is not required in the method body



```java
// Fig. 21.13: TotalNumbers.java
// Totaling the numbers in an ArrayList<Number>.
import java.util.ArrayList;

public class TotalNumbers
{
   public static void main( String[] args )
   {
      // create, initialize and output ArrayList of Numbers containing
      // both Integers and Doubles, then display total of the elements
      Number[] numbers = { 1, 2.4, 3, 4.1 }; // Integers and Doubles
      ArrayList< Number > numberList = new ArrayList< Number >();

      for ( Number element : numbers )
         numberList.add( element ); // place each number in numberList

      System.out.printf( "numberList contains: %s\n", numberList );
      System.out.printf( "Total of the elements in numberList: %.1f\n",
         sum( numberList ) );
   } // end main

   // calculate total of ArrayList elements
   public static double sum( ArrayList< Number > list )
   {
      double total = 0; // initialize total

      // calculate sum
      for ( Number element : list )
         total += element.doubleValue();

      return total;
   } // end method sum
} // end class TotalNumbers
```

```
numberList contains: [1, 2.4, 3, 4.1]
Total of the elements in numberList: 10.5
```

ArrayList<? Extends Number（超类）> 只要是 Number 的子类都是可以的

通配符扩展了 Number 类，他表示通配符的上界 Number，未知类型必须是 Number 的子类。

Public static<T extends Number> double sum(ArrayList<T> list);

通配函数

泛型的参数类型还可以是通配符类型。例如 Class<?> classType = Class.forName(java.lang.String);

81.sliders, menus, pop-up menus and windows

JMenuBar 管理菜单栏

JMenuItem 管理菜单单项

JCheckBoxMenuItem 单选菜单单项

JRadioButtonMenuItem

menu 和 popup menus 区别和使用代码：

JpopupMenu 必须要注册鼠标事件

```
Private checkForTriggerEvent(MouseEvent event)
{
    If(event.isPopupTrigger())
    {
        popupMenu.show(event.getComponent(),event.getX(),ev
    ent.getY());
    }
}
```

82.JDesktopPane and JInternalFrame

多文档界面（MDI）

a main window (called the parent window) containing other windows (called child windows), to manage several open documents that are being processed in parallel.

JDesktopPane 窗口/桌面面板和 JInternalFrame 子窗口组成了多文档界面

JInternalFrame 的构造函数使用了五个参数：

String：标题

boolean：能否拉伸

boolean：能否关闭

boolean：能否最大化

boolean：能否最小化

83.BoxLayout and GridBagLayout

1)BoxLayout 和 FlowLayout 有点类似，不过 BoxLayout 既可以横向排列组件，又设置为纵向排列组件，所以它的功能比 FlowLayout 强大。它总共有四种约束方式：X_AXIS，Y_AXIS，LINE_AXIS，PAGE_AXIS

X_AXIS：组件从左到右横向排列

Y_AXIS：组件从上到下纵向排列

LINE_AXIS：按照行的方式排列，可以从左到右也可以从右到左

PAGE_AXIS：按照页面的方式排列

new BoxLayout(container,BoxLayout.PAGE_AXIS)

2)GridBagLayout 比 GridLayout 相似但更灵活，组件可以有各种大小并且按照各种顺序添加

Field	Description
anchor	Specifies the relative position (NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST, CENTER) of the component in an area that it does not fill.
fill	Resizes the component in the specified direction (NONE, HORIZONTAL, VERTICAL, BOTH) when the display area is larger than the component.
gridx	The column in which the component will be placed.
gridy	The row in which the component will be placed.
gridwidth	The number of columns the component occupies.
gridheight	The number of rows the component occupies.
weightx	The amount of extra space to allocate horizontally. The grid slot can become wider when extra space is available.
weighty	The amount of extra space to allocate vertically. The grid slot can become taller when extra space is available.

Fig. 25.19 | GridBagConstraints fields.

84.How to create and execute a new thread

```java
// Fig. 26.3: PrintTask.java
// PrintTask class sleeps for a random time from 0 to 5 seconds
import java.util.Random;

public class PrintTask implements Runnable
{
   private final int sleepTime; // random sleep time for thread
   private final String taskName; // name of task
   private final static Random generator = new Random();

   // constructor
   public PrintTask( String name )
   {
      taskName = name; // set task name

      // pick random sleep time between 0 and 5 seconds
      sleepTime = generator.nextInt( 5000 ); // milliseconds
   } // end PrintTask constructor

```

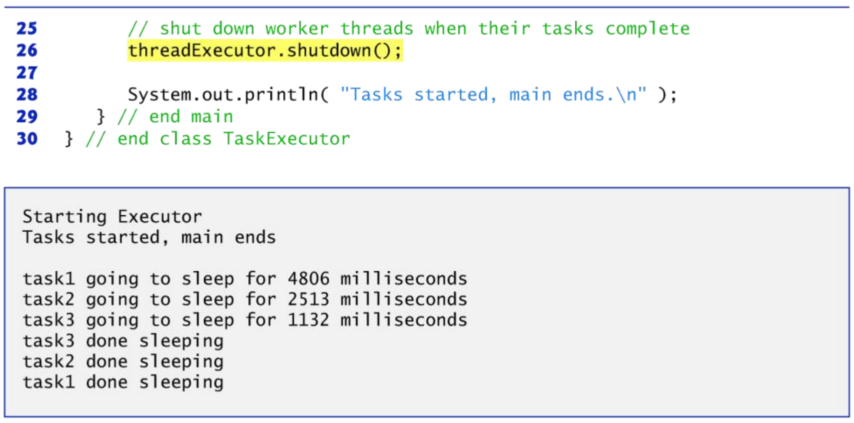
Fig. 26.3 | PrintTask class sleeps for a random time from 0 to 5 seconds. (Part 1 of 2.)

```java
    // method run contains the code that a thread will execute
    public void run()
    {
       try // put thread to sleep for sleepTime amount of time
       {
          System.out.printf( "%s going to sleep for %d milliseconds.\n",
             taskName, sleepTime );
          Thread.sleep( sleepTime ); // put thread to sleep
       } // end try
       catch ( InterruptedException exception )
       {
          System.out.printf( "%s %s\n", taskName,
             "terminated prematurely due to interruption" );
       } // end catch

       // print task name
       System.out.printf( "%s done sleeping\n", taskName );
    } // end method run
 } // end class PrintTask
```

Fig. 26.3 | `PrintTask` class sleeps for a random time from 0 to 5 seconds. (Part 2 of 2.)

```java
// Fig. 26.4: TaskExecutor.java
// Using an ExecutorService to execute Runnables.
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

public class TaskExecutor
{
   public static void main( String[] args )
   {
      // create and name each runnable
      PrintTask task1 = new PrintTask( "task1" );
      PrintTask task2 = new PrintTask( "task2" );
      PrintTask task3 = new PrintTask( "task3" );

      System.out.println( "Starting Executor" );

      // create ExecutorService to manage threads
      ExecutorService threadExecutor = Executors.newCachedThreadPool();

      // start threads and place in runnable state
      threadExecutor.execute( task1 ); // start task1
      threadExecutor.execute( task2 ); // start task2
      threadExecutor.execute( task3 ); // start task3

```

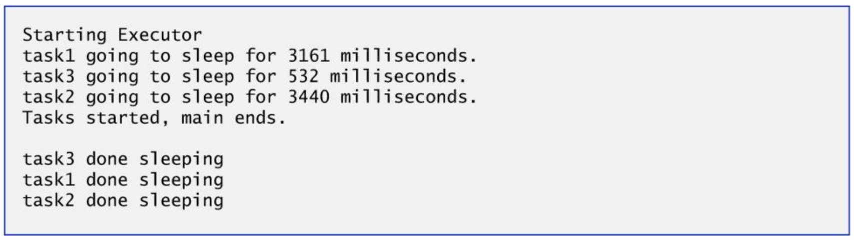
Fig. 26.4 | Using an `ExecutorService` to execute `Runnable`s. (Part 1 of 3.)



```java
        // shut down worker threads when their tasks complete
        threadExecutor.shutdown();

        System.out.println( "Tasks started, main ends.\n" );
     } // end main
  } // end class TaskExecutor
```

```
Starting Executor
Tasks started, main ends

task1 going to sleep for 4806 milliseconds
task2 going to sleep for 2513 milliseconds
task3 going to sleep for 1132 milliseconds
task3 done sleeping
task2 done sleeping
task1 done sleeping
```

Fig. 26.4 | Using an `ExecutorService` to execute `Runnables`. (Part 2 of 3.)



```
Starting Executor
task1 going to sleep for 3161 milliseconds.
task3 going to sleep for 532 milliseconds.
task2 going to sleep for 3440 milliseconds.
Tasks started, main ends.

task3 done sleeping
task1 done sleeping
task2 done sleeping
```

Fig. 26.4 | Using an `ExecutorService` to execute `Runnables`. (Part 3 of 3.)

85.The life cycle of a thread

new state, waiting state, timed waiting state, blocked state, terminated state

86.Thread synchronization

Provided to the programmer with mutual exclusion

Exclusive access to a shared object， during that time, other threads desiring to access the object are kept waiting.

Implemented in Java using locks

87. To implement Java networking application by using sockets and datagrams

With datagram sockets, individual packets of information are transmitted

UDP—the User Datagram Protocol—is a connectionless service, and thus does not guarantee that packets arrive in any particular order.

Packets can even be lost or duplicated

88. Connect to a database

```
// Fig. 25.25: DisplayAuthors.java
// Displaying the contents of the authors table.
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

public class DisplayAuthors
{
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DATABASE_URL = "jdbc:mysql://localhost/books";

   // launch the application
   public static void main( String args[] )
   {
      Connection connection = null; // manages connection
```

```java
      Statement statement = null; // query statement

      // connect to database books and query database
      try
      {
         Class.forName( JDBC_DRIVER ); // load database driver class

         // establish connection to database
         connection =
            DriverManager.getConnection( DATABASE_URL, "jhtp6", "jhtp6" );

      catch ( ClassNotFoundException classNotFound )
      {
         classNotFound.printStackTrace();
         System.exit( 1 );
      } // end catch
      finally // ensure statement and connection are closed properly
      {
         try
         {
            statement.close();
            connection.close();
         } // end try
         catch ( Exception exception )
         {
            exception.printStackTrace();
            System.exit( 1 );
         } // end catch
      } // end finally
   } // end main
} // end class DisplayAuthors
```

```
Authors Table of Books Database:
authorID        firstName       lastName
1               Harvey           Deitel
2               Paul            Deitel
3               Tem              Nieto
4               Sean             Santry
```