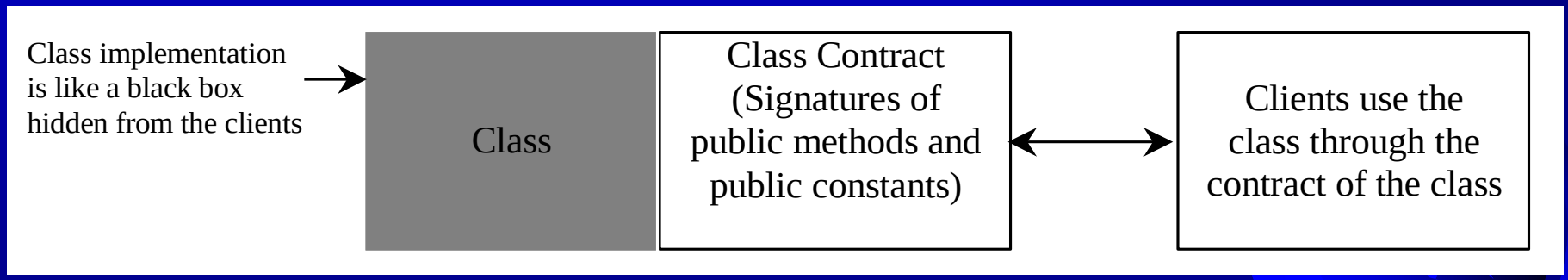
Chapter 10 Object-Oriented Thinking

类抽象和封装

- 类抽象指的是将一个类的具体实现和类的使用做一个分离。
- 作为类的创建者，他只需要提供这个类的描述，以便让使用者知道如何使用这个类；
- 作为类的使用者，他不需要知道这个类的内部如何实现，也无从知道这个类如何实现。



看一个例子来感受类抽象

- 显然，即便你不知道如何Stage类是怎么做出来的，不妨碍你可以使用它做一个自己的界面。

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
public class MyJavaFX extends Application {
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    pane.getChildren().add(new Button("OK"));
    Scene scene = new Scene(pane, 400, 300);
    primaryStage.setTitle("My first JavaFX");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
}
```

面向对象的思维方式

- 面向对象非常适合开发大型程序，因为它对代码重用的支持是非常好的。
- 当然如果是小型程序，或者你根本不考虑代码重用的问题，那么面向过程的思维方式就足够了。
- 来看一个计算身体质量指数的例子。

计算身体质量指数

- Body Mass Index (BMI)是利用身高和体重计算是否健康的一个指数，公式为：**BMI=体重（kg）÷ 身高（m）^2**
- 利用上述公式计算某人的BMI指数并给出解释。

BMI	Interpretation
Below 18.5	Underweight
18.5–24.9	Normal
25.0–29.9	Overweight
Above 30.0	Obese

```java
import java.util.Scanner;

public class ComputeAndInterpretBMI {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter weight in pounds
    System.out.print("Enter weight in pounds: ");
    double weight = input.nextDouble();

    // Prompt the user to enter height in inches
    System.out.print("Enter height in inches: ");
    double height = input.nextDouble();

    final double KILOGRAMS_PER_POUND = 0.45359237; // Constant
    final double METERS_PER_INCH = 0.0254; // Constant

    // Compute BMI
    double weightInKilograms = weight * KILOGRAMS_PER_POUND;
    double heightInMeters = height * METERS_PER_INCH;
    double bmi = weightInKilograms /
      (heightInMeters * heightInMeters);

    // Display result
    System.out.println("BMI is " + bmi);
    if (bmi < 18.5)
      System.out.println("Underweight");
    else if (bmi < 25)
      System.out.println("Normal");
    else if (bmi < 30)
      System.out.println("Overweight");
    else
      System.out.println("Obese");
  }
}
```

- 如果不用面向对象的思维方式，可以写出左边的代码。
- 这段代码的重用性很差。
- 例如，想把程序升级成有界面的版本，只能把计算BMI部分的代码先找出来，然后我贴，我贴，我贴贴贴……

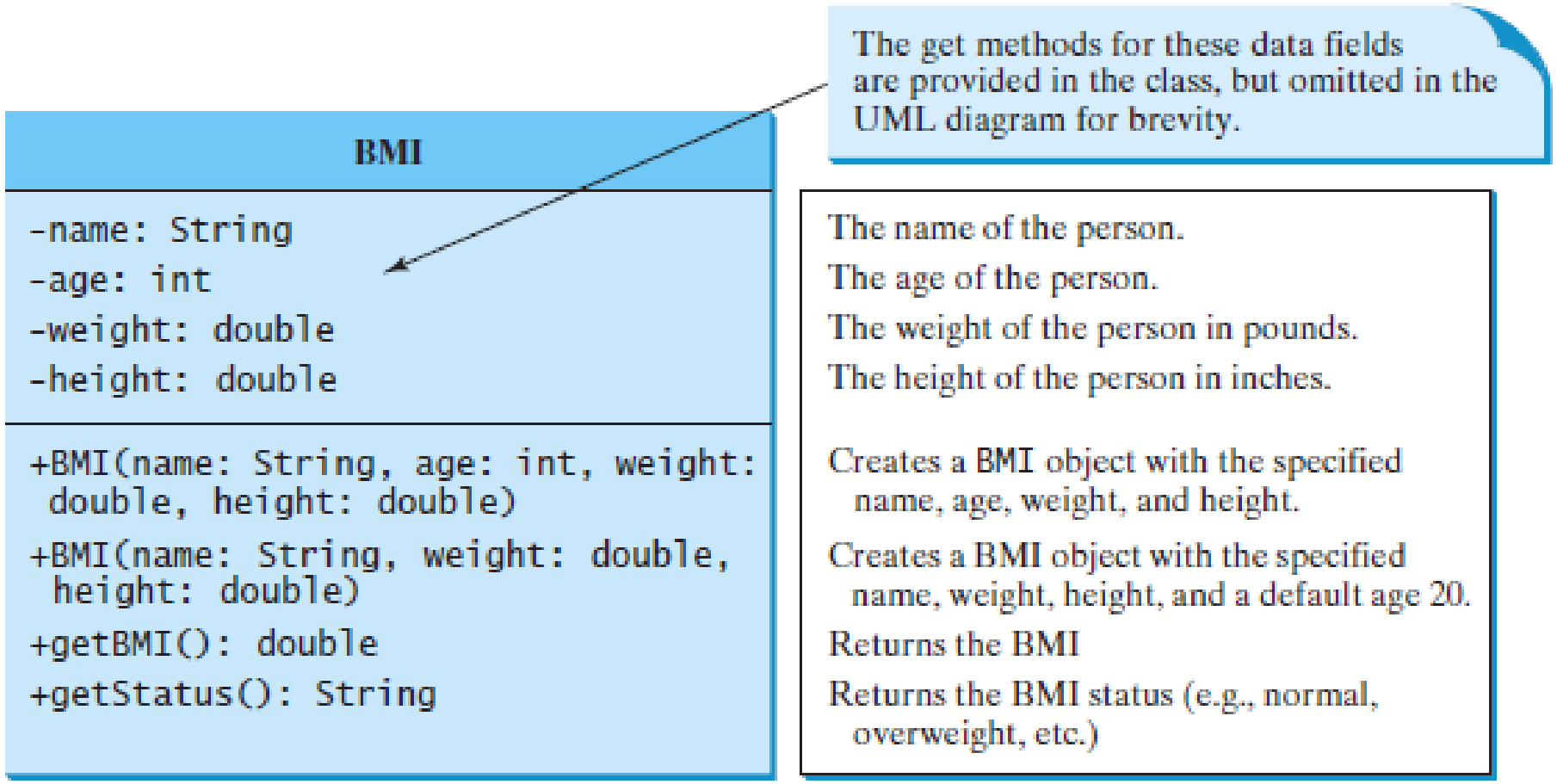
改进一点

- 把BMI计算单独写成一个方法：

```
public static double getBMI(double weight,
    double height)
```

- 不过这样依然有所不足，例如你想同时用变量存储一个人的姓名和生日，包括他的BMI，这时候这几个变量就完全是松散的，无法关联在一起。为了让它们成为一个整体，只能用面向对象的技术将BMI包装成一个类。

BMI的UML图


BMI
-name: String
-age: int
-weight: double
-height: double
+BMI(name: String, age: int, weight: double, height: double)
+BMI(name: String, weight: double, height: double)
+getBMI(): double
+getStatus(): String
The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The name of the person.
The age of the person.
The weight of the person in pounds.
The height of the person in inches.
Creates a BMI object with the specified name, age, weight, and height.
Creates a BMI object with the specified name, weight, height, and a default age 20.
Returns the BMI
Returns the BMI status (e.g., normal, overweight, etc.)

BMI.java－1/3

```
public class BMI {
private String name;
private int age;
private double weight; // in pounds
private double height; // in inches
public static final double KILOGRAMS_PER_POUND = 0.45359237;
public static final double METERS_PER_INCH = 0.0254;

public BMI(String name, int age, double weight, double height) {
  this.name = name;
  this.age = age;
  this.weight = weight;
  this.height = height;
}

public BMI(String name, double weight, double height) {
  this(name, 20, weight, height);
}
```

BMI.java－2/3

```
public double getBMI() {
  double bmi = weight * KILOGRAMS_PER_POUND
/ ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
  return Math.round(bmi * 100) / 100.0;
}

public String getStatus() {
  double bmi = getBMI();
  if (bmi < 18.5)
    return "Underweight";
  else if (bmi < 25)
    return "Normal";
  else if (bmi < 30)
    return "Overweight";
  else
    return "Obese";
}
```

BMI.java－3/3

```
public String getName() {
  return name;
}

public int getAge() {
  return age;
}

public double getWeight() {
  return weight;
}

public double getHeight() {
  return height;
}
}
```

UseBMIClass.java

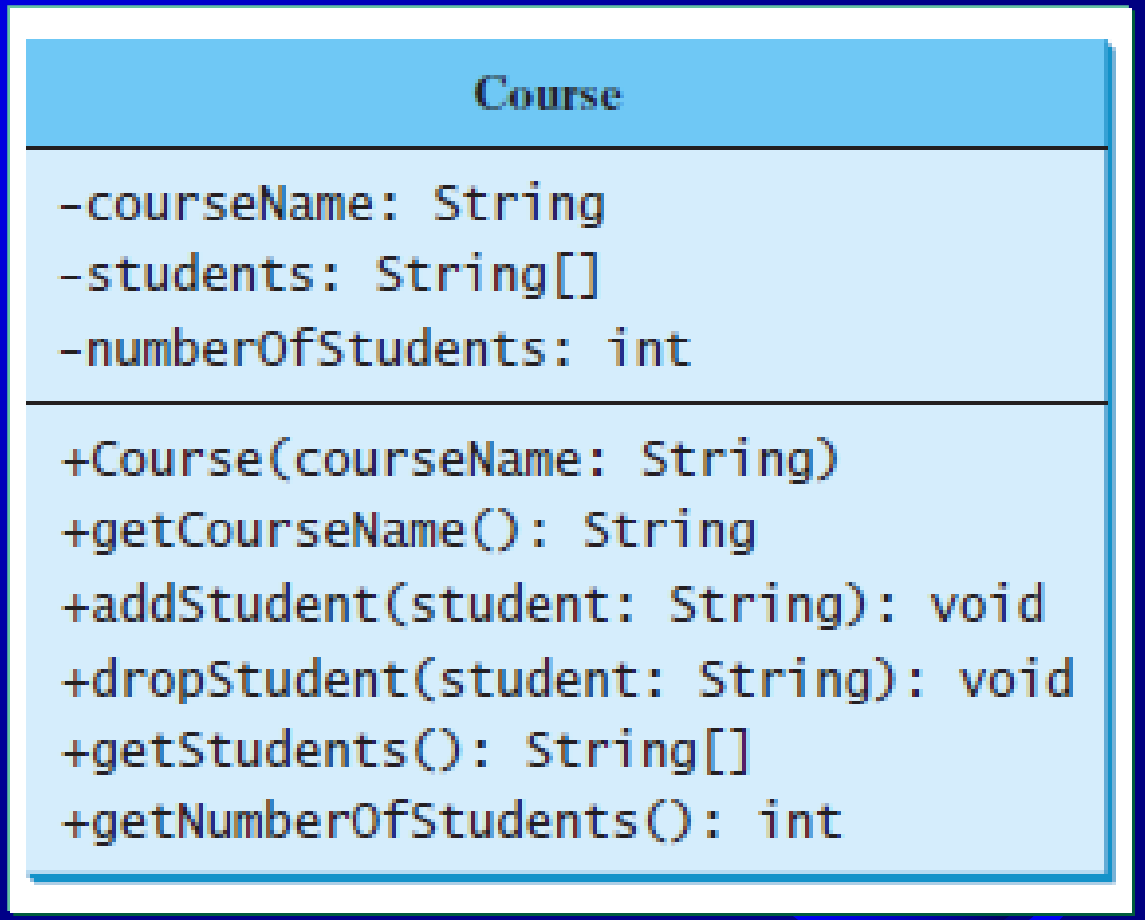
```
public class UseBMIClass {
public static void main(String[] args) {

BMI bmi1 = new BMI("Kim Yang", 18, 145, 70);
System.out.println("The BMI for " + bmi1.getName()
+ " is " + bmi1.getBMI() + " " + bmi1.getStatus());

BMI bmi2 = new BMI("Susan King", 215, 70);
System.out.println("The BMI for " + bmi2.getName()
+ " is " + bmi2.getBMI() + " " + bmi2.getStatus());

}
}
```

小结

- 面向过程侧重于考虑**方法的编写**，面向对象则致力于将数据和方法先做一个封装，侧重于考虑**对象和对象的操作**。
- 面向过程的编程，数据和对数据的操作是分开的，造成了数据需要在不同方法中反复传递；面向对象很好地反映了真实的世界中，所有的对象都是相关的属性和活动。使用对象，提高了软件的可重用性并使得程序更容易开发和维护。一个Java程序可以看作是一堆合作的对象的集合。

编程练习：课程类

- 属性设计：
  - 课程名
  - 选课名单
  - 选课人数
- 方法设计：
  - 构造方法
  - 返回课程姓名
  - 添加学生
  - 返回选课学生
  - 返回选课人数



```
public class Course {
private String courseName;
private String[] students = new String[100];
private int numberOfStudents;

public Course(String courseName) {
  this.courseName = courseName;
}
public void addStudent(String student) {
  students[numberOfStudents] = student;
  numberOfStudents++;
}
public String[] getStudents() {
  return students;
}
public int getNumberOfStudents() {
  return numberOfStudents;
}
public String getCourseName() {
  return courseName;
}
}
```

```
public class TestCourse {
public static void main(String[] args) {
Course course1 = new Course("Data Structures");
Course course2 = new Course("Database Systems");

course1.addStudent("Peter Jones");
course1.addStudent("Kim Smith");
course1.addStudent("Anne Kennedy");

course2.addStudent("Peter Jones");
course2.addStudent("Steve Smith");

System.out.println("Number of students in course1: "
+ course1.getNumberOfStudents());
String[] students = course1.getStudents();
for (int i = 0; i < course1.getNumberOfStudents(); i++)
  System.out.print(students[i] + ", ");

System.out.print("Number of students in course2: "
+ course2.getNumberOfStudents());
}
}
```

小结

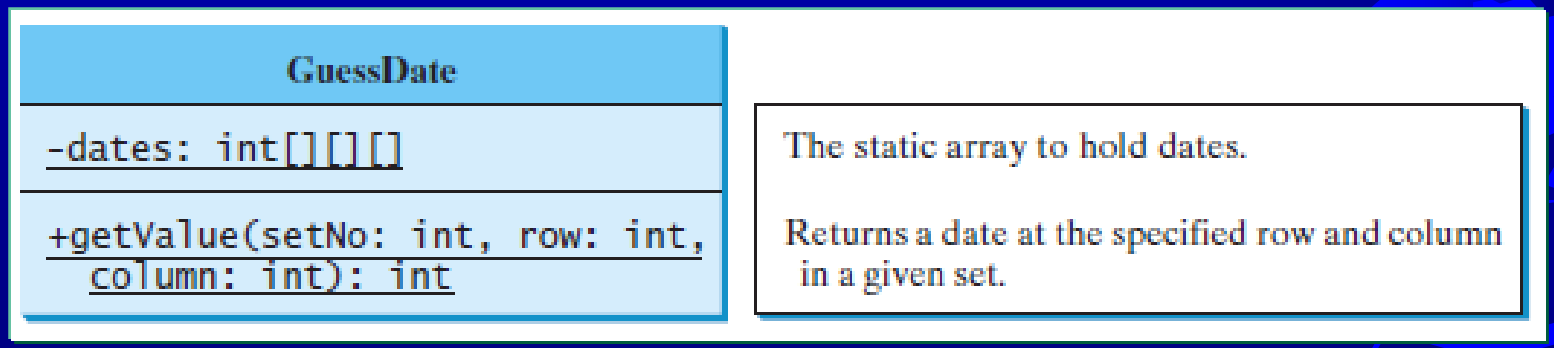
- 为了简单，课程类的学生数上限是100。
- 创建一个**Course**对象之后，学生数组同时被创建，**Course**其实仅仅记录了这个数组的引用，严格来说，数组并不在**Course**的存储空间内，不过你可以简单这么认为。
- 对类的使用者而言，他只需要借助**addStudent**, **getNumberOfStudents**等方法即可使用这个类，完全不用关心这个类的内部如何实现。例如他根本不必知道学生姓名如何被存储，是数组、链表或是其它数据结构。

设计GuessDate类

- 还记得猜生日的程序吧？前后我们写过2个版本，都用到了同样的5张表。假如我们要再写一个更漂亮的界面来猜生日，毫无疑问又要把5张表的数据再贴到现在的代码中。
- 有没有办法可以做到，5个表的数据单独做好，任何程序都可以借用？这样就不用反复贴一堆数字了。答案是，面向对象，我们把5个表的数字单独做一个类。

GuessDate类的设计

- 由于5个表的数据是常量，无论在哪个程序中，这些数据都是一样的，所以没有必要把这个类设计成实例相关，因此把所有数据设置成final static是最好的。
- 用户无需知道这些数据的存储细节，因此只需要提供一个getValue的方法就行。



```
public class GuessDate {
private final static int[][][] dates = {
{ { 1, 3, 5, 7 }, { 9, 11, 13, 15 }, { 17, 19, 21, 23 },
{ 25, 27, 29, 31 } },
{ { 2, 3, 6, 7 }, { 10, 11, 14, 15 }, { 18, 19, 22, 23 },
{ 26, 27, 30, 31 } },
{ { 4, 5, 6, 7 }, { 12, 13, 14, 15 }, { 20, 21, 22, 23 },
{ 28, 29, 30, 31 } },
{ { 8, 9, 10, 11 }, { 12, 13, 14, 15 }, { 24, 25, 26, 27 },
{ 28, 29, 30, 31 } },
{ { 16, 17, 18, 19 }, { 20, 21, 22, 23 }, { 24, 25, 26, 27 },
{ 28, 29, 30, 31 } } };

/** 不让用户使用类似 GuessDate g = new GuessDate(); 这样的语法 */
private GuessDate() {
}
/** 返回指定位置的数字，参数是：表序号，行，列 */
public static int getValue(int setNo, int i, int j) {
  return dates[setNo][i][j];
}
}
```

```java
import java.util.Scanner;

public class UseGuessDateClass {
  public static void main(String[] args) {
    int date = 0; // Date to be determined
    int answer;

    // Create a Scanner
    Scanner input = new Scanner(System.in);

    for (int i = 0; i < 5; i++) {
      System.out.println("Is your birthday in Set" + (i + 1) + "?");
      for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 4; k++)
          System.out.print(GuessDate.getValue(i, j, k) + "  ");
        System.out.println();
      }

      System.out.print("\nEnter 0 for No and 1 for Yes: ");
      answer = input.nextInt();

      if (answer == 1)
        date += GuessDate.getValue(i, 0, 0);
    }

    System.out.println("Your birthday is " + date);
  }
}
```

小结

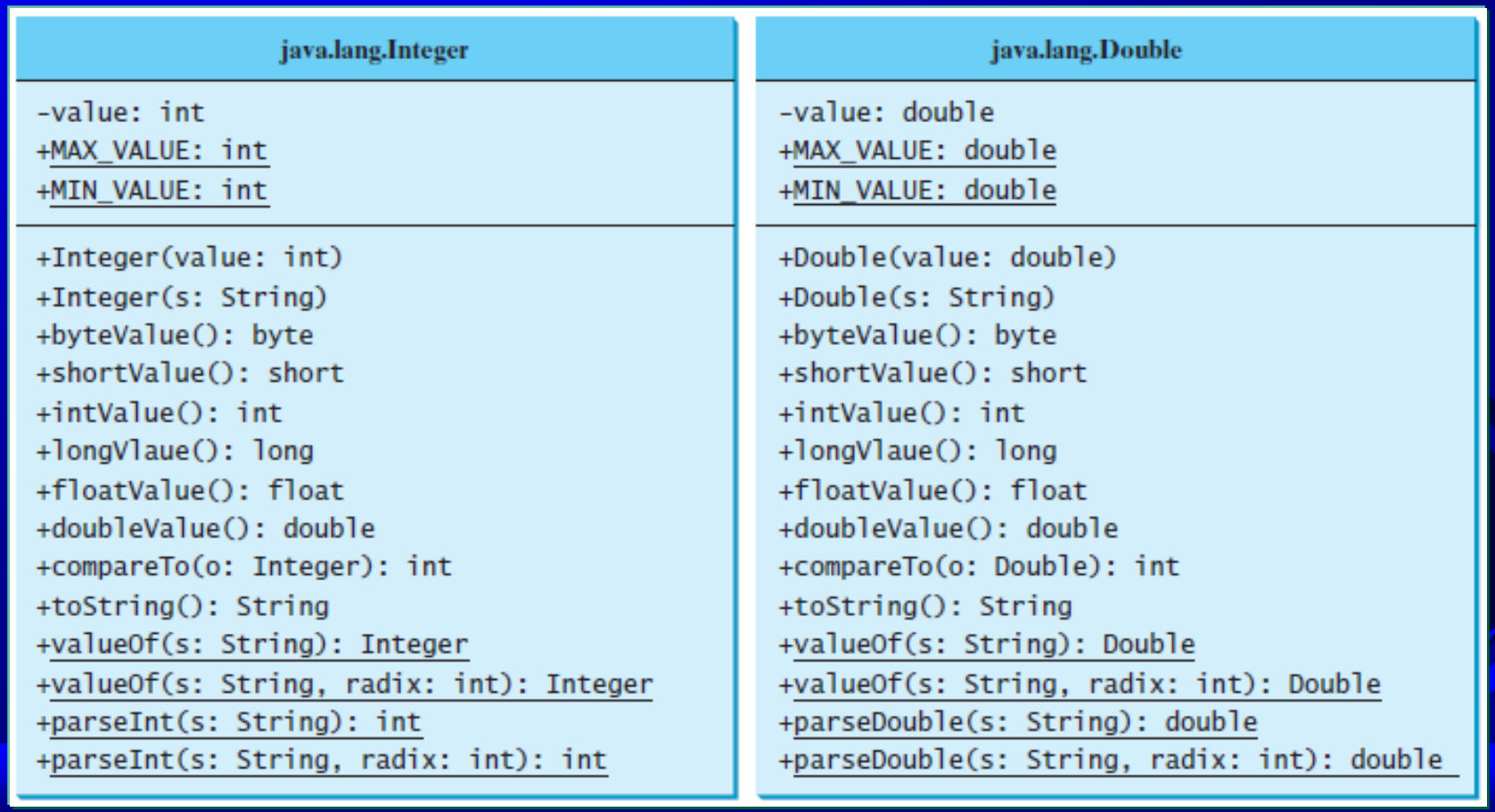
- 这个GuessDate类使用了三维数组存储5张表，如果哪天你心血来潮想用5个二维数组来存储并修改了GuessDate类，只要保持getValue方法不变，重编译GuessDate即可。UseGuessDateClass这个程序完全不用修改，这就是封装性的好处。
- 把GuessDate方法设置为private，是因为它的成员都是static的，把这种类new出来并没有多大意义，干脆禁止用户这么使用。

用对象的方式处理基本类型

- OOP号称“Everything is an Object”，所以Java中唯一美中不足的，就是那些基本类型(char, int, float, double...)了。
- 为了弥补这个缺憾，Java为每一种基本类型设计了对应的包装类。功能上它们能够完全取代基本类型，当然，用起来要麻烦一点。

数值型包装类

- 数值型包装类Integer和Double的成员如图，注意有下划线的是**静态成员**。



数值型包装类的构造方法

- 可以直接从基本数值类型（或形如数值的字符串），作为参数去构造出一个对应的包装类。
- 例如：
  - Double doubleObject = new Double(5.0);
  - Double doubleObject = new Double("5.0");
  - Integer integerObject = new Integer(5);
  - Integer integerObject = new Integer("5");
- 在JDK1.5以上版本，你还可以这样偷懒一把：
  - Double doubleObject = 5.0;
  - Integer integerObject = 5;

数值型包装类的常量

- 数值型包装类提供了本类型的最大值和最小值的**静态常量** MAX_VALUE 和MIN_VALUE，你可以从两个数得知该类型的取值范围。
- 例如：
  - System.out.println("The maximum integer is " + Integer.MAX_VALUE);
  - System.out.println("The minimum positive float is " + Float.MIN_VALUE);
  - System.out.println("The maximum double precision floating-point number is " + Double.MAX_VALUE);

数值型包装类 => 基本类型

- 可以使用包装类的doubleValue, floatValue, intValue, longValue, shortValue函数，从包装类取出对应的基本类型的值。
- 例如：
  - long l = doubleObject.longValue(); //有截断
  - int i = integerObject.intValue();

数值包装类和字符串的转换

- 数值包装类 => 字符串
  - double d = 5.9;
  - Double doubleObject = new Double(d);
  - String s = doubleObject.toString();
- 字符串 => 数值包装类
  - Double doubleObject = Double.valueOf("1.4");
  - Integer integerObject = Integer.valueOf("12");

字符串 => 基本类型

- // These two methods are in the Byte class
- public static byte parseByte(String s)
- public static byte parseByte(String s, int radix)

- // These two methods are in the Short class
- public static short parseShort(String s)
- public static short parseShort(String s, int radix)

- // These two methods are in the Integer class
- public static int parseInt(String s)
- public static int parseInt(String s, int radix)

- 例如： **Integer.parseInt("11", 2)** returns **3**;
  Integer.parseInt("1A", 16) returns **26**;

字符串 => 基本类型

- // These two methods are in the Long class
- public static long parseLong(String s)
- public static long parseLong(String s, int radix)

- // These two methods are in the Float class
- public static float parseFloat(String s)
- public static float parseFloat(String s, int radix)

- // These two methods are in the Double class
- public static double parseDouble(String s)
- public static double parseDouble(String s, int radix)

BigInteger和BigDecimal

- 这两个类用于处理任意精度的整数和浮点数运算，不会有计算溢出问题。
- 不过这两个类不能使用普通的运算符（例如：+-*/%），只能用add, subtract, multiply, divide和remainder等成员方法，和另一个类对象做运算。
- 看两个例子。

例题一

```
BigInteger a = new BigInteger("92233720368547758 07");
BigInteger b = new BigInteger("2");
BigInteger c = a.multiply(b); // 92233720368547758 07 * 2
System.out.println(c); // 184467440737095516 14

BigDecimal a = new BigDecimal(1.0);
BigDecimal b = new BigDecimal(3);
BigDecimal c = a.divide(b, 20, BigDecimal.ROUND_UP);
System.out.println(c); //0.33333333333333333334
```

例题二

LISTING 10.9 LargeFactorial.java

```
1  import java.math.*;
2
3  public class LargeFactorial {
4    public static void main(String[] args) {
5      System.out.println("50! is \n" + factorial(50));
6    }
7
8    public static BigInteger factorial(long n) {
9      BigInteger result = BigInteger.ONE;                    constant
10     for (int i = 1; i <= n; i++)
11       result = result.multiply(new BigInteger(i + ""));    multiply
12
13     return result;
14   }
15 }
```

```
50! is
30414093201713378043612608166064768844377641568960512000000000000
```

构造一个字符串

- 标准的字符串构造方法：

String newString = new String(stringLiteral);

String message = new String("Welcome to Java");

- 由于字符串用的太频繁了，所以Java提供了一个简写形式的构造方式：

String message = "Welcome to Java";

- 注意：其实，上述两种方法并不完全等价，它们构造出来的字符串，其**存储原理**还是有细微差别的。不过构造出来之后，用法是一样的。

字符串内容是不可变的

- 字符串一旦创建完毕，其内容是不可变的。换句话说，字符串的内容是常量。不过下面这个例子可能有些误导，"Java"字符串的内容究竟被改变了没有？

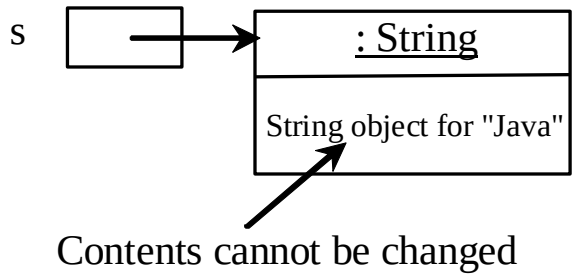
```
String s = "Java";
s = "HTML";
```

- 结论是显然的：内容未变。那么被改变的是什么？答案是s的指向，或者说s所引用的字符串变成另外一个串了。

单步执行一下

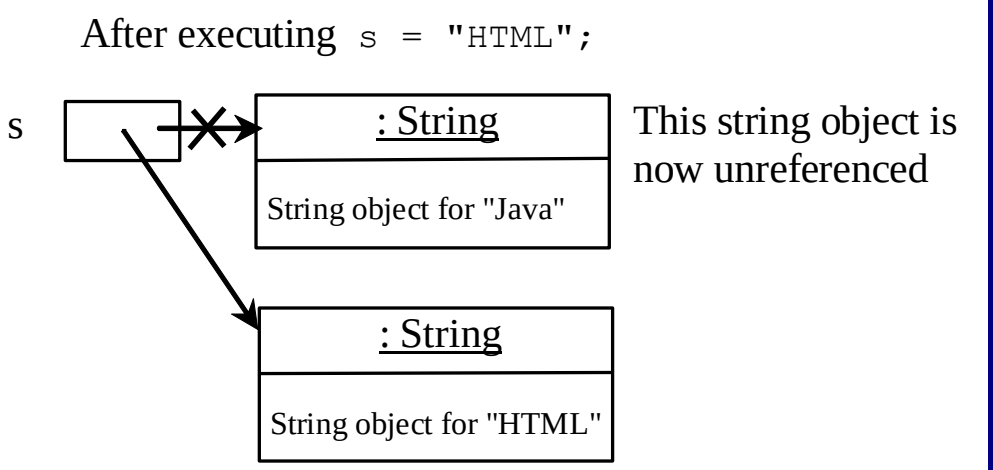
```
String s = "Java";
s = "HTML";
```

After executing `String s = "Java";`



单步执行一下

```
String s = "Java";
s = "HTML";
```

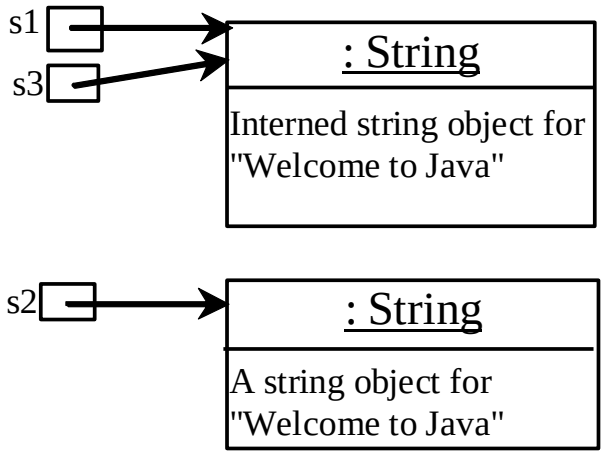


驻留字符串（Interned Strings）

- 由于字符串内容不可更改，因此JVM为了节约内存和提高效率，对于相同内容的字符串，JVM在内存中只留一份，然后大家共享。
- 这种只留一份大家共享的字符串，称为驻留字符串（Interned Strings）。你可以理解为，在没有使用new，采用类似（String s = "Java"; ）这种方式创建字符串的情况下，Java不会让相同内容的字符串在内存中出现2次。例如：

一个例子

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```



输出结果：

s1 == s2 is false

s1 == s3 is true

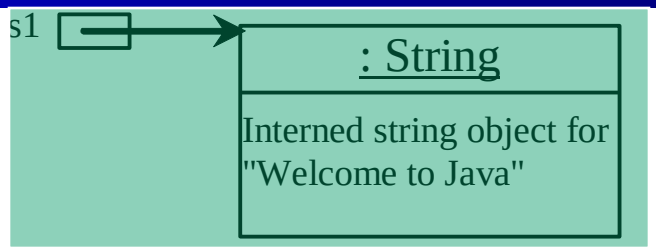
注意==运算符比较的不是字符串内容，而是是否引用同一对象。

结论：

- new运算符能确保String新建一个字符串；
- 如果采用简写方式创建字符串，JVM会采用驻留字符串机制，尽可能共享内容相同的字符串。

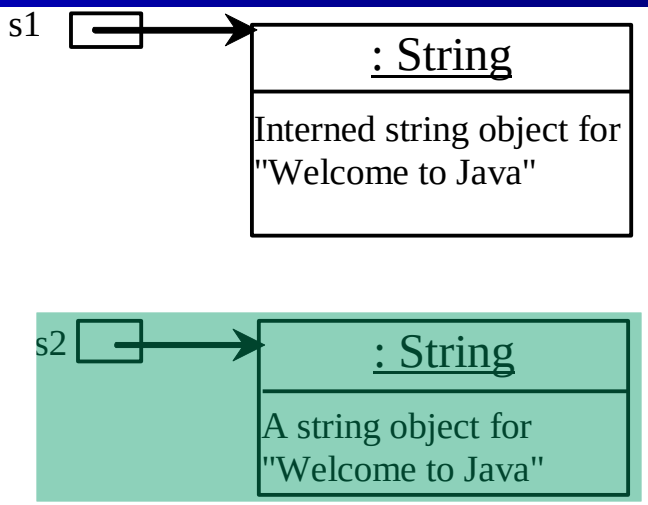
单步执行一下

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```



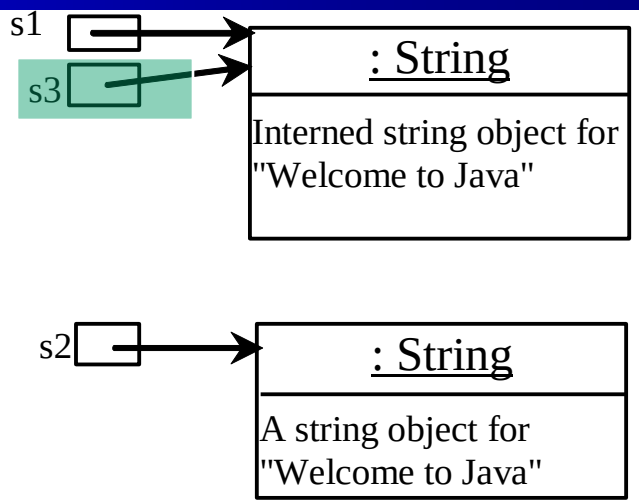
单步执行一下

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```



单步执行一下

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```



字符串内容相等比较

- equals

```
String s1 = new String("Welcome");
String s2 = "welcome";

if (s1.equals(s2)){
  //比较s1，s2的内容是否相等
}
if (s1 == s2) {
 //比较s1，s2是否相同引用（即指向同一个对象）
}
```

字符串比较大小

- compareTo(Object object)

```
String s1 = new String("Welcome");
String s2 = "welcome";

if (s1.compareTo(s2) > 0) {
   // s1 大于 s2
}
else if (s1.compareTo(s2) == 0) {
   // s1 等于 s2
}
else
    // s1 小于 s2
```

字符串长度

可以通过长度函数得到字符串长度（注意Java的字符是双字节的，所以单个汉字或英文字母长度都算1）：

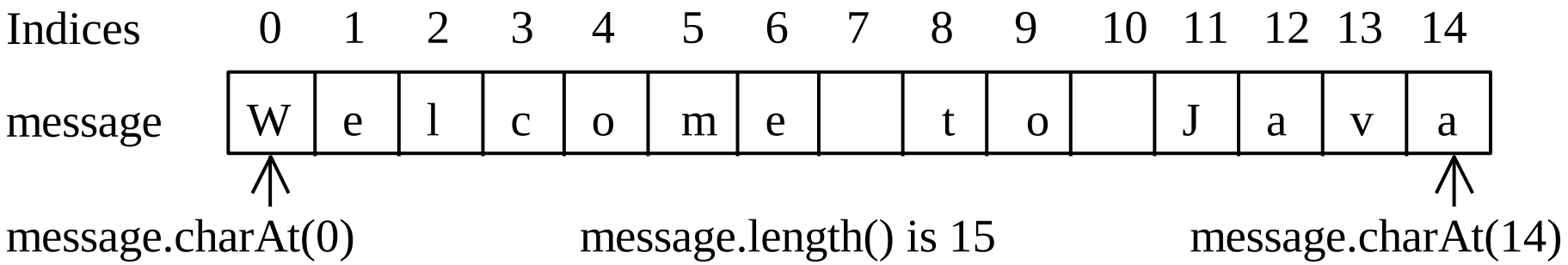
```
message = "Welcome";
message.length() (返回 7)
```

注意字符串长度有小括号()，这点和数组不同。数组的长度是属性：

int[] myList=new int[5]; myList.length才是长度

访问字符串的单个字符

- 不要使用下标方式，如：`message[0]`
- 必须使用：`message.charAt(index)`
- 老习惯，下标从0开始



字符串拼接

```
String s3 = s1.concat(s2);
```

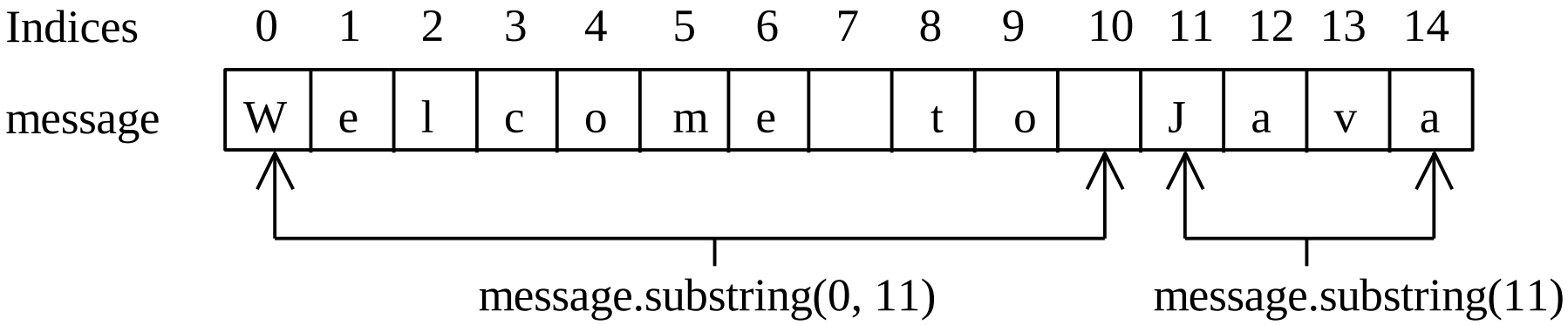
String s3 = s1 + s2;

s1 + s2 + s3 + s4 + s5 等价于
(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

子串抽取

使用 substring 方法可以从字符串中抽取子串，该方法的有两个重载版本，一是substring(beginIndex: int)，表示从起始下标抽取到串尾；二是substring(beginIndex: int, endIndex: int)，表示从起始下标抽取到终止下标，但不含终止下标那个字符（即抽取到endIndex-1）。

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0, 11) + "HTML";
```



字符串替换函数举例

"Welcome". toLowerCase() 返回新串 welcome.

"Welcome".toUpperCase()返回新串 WELCOME.

" Welcome ".trim()返回新串 Welcome.

"Welcome".replace('e', 'A')返回新串 WAlcomA.

"Welcome".replaceFirst("e", "AB")返回新串WABlcome.

"Welcome".replace("e", "AB")返回新串 WABlcomAB.

"Welcome".replace("el", "AB")返回新串 WABlcome.

注意：因为字符串内容不可变，上述操作都是通过另外新建一个字符串的方式实现。

字符串拆分split

```
String[] tokens = "Java#HTML#Perl".split("#");
for (int i = 0; i < tokens.length; i++)
  System.out.println("No." + (i + 1) + " " +
tokens[i]);
```

运行结果是:

No.1 Java

No.2 HTML

No.3 Perl

使用正则表达式拆分字符串

上述例子如果单词之间的#数是不定的，普通拆分就搞不定了，例如下面这个例子：

```
String[] tokens = "Java##HTML###Perl".split("#");
for (int i = 0; i < tokens.length; i++)
  System.out.println("No." + (i + 1) + " " + tokens[i]);
```

运行结果是：

No.1 Java

No.2

No.3 HTML

No.4

No.5

No.6 Perl

使用正则表达式拆分字符串

采用正则表达式做参数可以解决上面那个问题：

```
String[] tokens = "Java##HTML###Perl".split("[#]+");
for (int i = 0; i < tokens.length; i++)
  System.out.println("No." + (i + 1) + " " + tokens[i]);
```

运行结果是：

No.1 Java

No.2 HTML

No.3 Perl

☞P.S. 关于正则表达式，这是个很大的话题了。有兴趣可以自行学习。这里我们只需要记住String支持使用正则表达式进行拆分就可以了。

字符或子串查找

"Welcome to Java".indexOf('W') 返回 0.

"Welcome to Java".indexOf('x')返回 -1.

"Welcome to Java".indexOf('o', 5)返回 9.

"Welcome to Java".indexOf("come")返回 3.

"Welcome to Java".indexOf("Java", 5)返回 11.

"Welcome to Java".indexOf("java", 5)返回 -1.

"Welcome to Java".lastIndexOf('a')返回 14.

字符串转成字符数组

String和字符数组本质上是不同的对象，不过它们之间可以互相转换。例如字符串可以这样转成字符数组：

```
char[] chars = "Java".toCharArray();
```

转完之后，

chars[0]是J, chars[1]是a, chars[2]是v, chars[3]是a

字符数组转成字符串

字符数组转成字符串可以这样做：

```
char[] arr = {'J', 'A', 'V', 'A'};
String str = String.valueOf(arr);
```

其实valueOf还能用于将char, double, long, int, float, boolean等类型转成字符串，例如：

```
String s = String.valueOf(5.44); //s变为"5.44"
```

当然你也可以这样转： **String s = ""+5.44;**

编程判断字符串是否回文

- 解题思路：分别从字符串的一头一尾开始，比较所在字符是否相等，然后各自前进一步继续判断，直到相遇或者所在字符不等为止。

判断回文的函数

```
/** Check if a string is a palindrome */
public static boolean isPalindrome(String s) {
  // The index of the first character in the string
  int low = 0;

  // The index of the last character in the string
  int high = s.length() - 1;

  while (low < high) {
    if (s.charAt(low) != s.charAt(high))
      return false; // Not a palindrome

    low++;
    high--;
  }

  return true; // The string is a palindrome
}
```

StringBuilder 和StringBuffer

- 这两个类是String的替代类，你可以在任何使用String类的地方改用这两个类。
- 这两个类的功能显然比String更强大，因为它们可以**就地**添加、插入和修改字符串的内容。
- 二者的区别是，StringBuilder是同步的，每次只能被一个线程修改；StringBuffer则允许多个线程同时修改同一个字符串。
- 如果你的程序是单线程的，不需要同时修改字符串，使用StringBuilder的效率会更高。

几个例子

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append(“Java”);
//现在字符串是Welcome to Java

stringBuilder.insert(11, "HTML and ");
//现在字符串是Welcome to HTML and Java.
```

假设stringBuilder内容是Welcome to Java，以下操作的效果分别是：

- stringBuilder.delete(8, 11); //变为Welcome Java.
- stringBuilder.deleteCharAt(8); //变为Welcome o Java.
- stringBuilder.reverse(); //变为avaJ ot emocleW.
- stringBuilder.replace(11, 15, "HTML"); //变为Welcome to HTML.
- stringBuilder.setCharAt(0, 'w'); //变为welcome to Java.

编程练习：检查字符串是否回文，非数字或者非字母的字符忽略不计

例如： ab<c>cb?a是回文，因为去掉非字母的剩下字符是： abccba

abcc><?cab不是回文，因为过滤后剩下的字符是：abcccab

解题思路：先将字符串过滤，生成新串；再比较新串和它的反转串是否相等。

回文判断函数

```
/** Return true if a string is a palindrome */
public static boolean isPalindrome(String s) {
// Create a new string by eliminating
nonalphanumeric chars
String s1 = filter(s);

// Create a new string that is the reversal of s1
String s2 = reverse(s1);

// Check if the reversal is the same as the
original string
return s2.equals(s1);
}
```

过滤非数字、非字母的字符

```
/** Create a new string by eliminating
nonalphanumeric chars */
public static String filter(String s) {
// Create a string builder
StringBuilder stringBuilder = new StringBuilder();
// Examine each char in the string to skip
alphanumeric char
for (int i = 0; i < s.length(); i++) {
  if( Character.isLetterOrDigit(s.charAt(i))) {
    stringBuilder.append(s.charAt(i));
  }
}
// Return a new filtered string
return stringBuilder.toString();
}
```

字符串反转函数

```
/** Create a new string by reversing a specified
string */
public static String reverse(String s) {
StringBuilder stringBuilder = new StringBuilder(s);

// Invoke reverse in StringBuilder
stringBuilder.reverse();

return stringBuilder.toString();
}
```

THE END