Introduction to computer programming A
LAB7

贾艳红 Jana
Email:jiayh@mail.sustech.edu.cn

LAB OBJECTIVES

- Learn how to define a Java class and create its object
- Learn how to define and use instance variables
- Learn how to define and use instance methods
- Learn how to use get and set methods
- Learn how to use ArrayList and make the object as its element.

knowledge points

Define a Java class and create its object

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created.

Here's how a class is defined:

```java
class ClassName {
  // variables
  // methods
}
```

Define a Java class and create its object

For example : Defining a class named Circle and its three objects.

```java
// Define the circle class with two constructors
class SimpleCircle {
  double radius;

  /** Construct a circle with radius 1 */
  SimpleCircle() {
    radius = 1;
  }

  /** Construct a circle with a specified radius */
  SimpleCircle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }

  /** Return the perimeter of this circle */
  double getPerimeter() {
    return 2 * radius * Math.PI;
  }

  /** Set a new radius for this circle */
  void setRadius(double newRadius) {
    radius = newRadius;
  }
}
```


SimpleCircle
radius: double
SimpleCircle()
SimpleCircle(newRadius: double)
getArea(): double
getPerimeter(): double
setRadius(newRadius: double): void
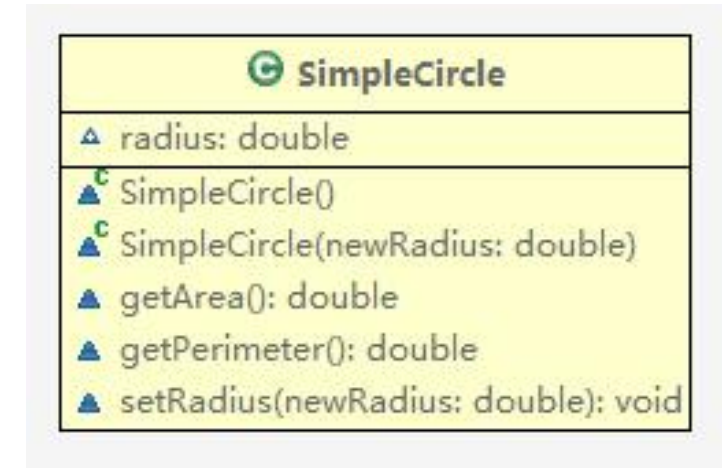
class diagram

● **How to Define Classes**

Define a Java class and create its object

For example : Defining a class named Circle and its three objects.

```java
public class TestSimpleCircle {
  /** Main method */
  public static void main(String[] args) {
    // Create a circle with radius 1
    SimpleCircle circle1 = new SimpleCircle();
    System.out.println("The area of the circle of radius "
      + circle1.radius + " is " + circle1.getArea());

    // Create a circle with radius 25
    SimpleCircle circle2 = new SimpleCircle(25);
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());

    // Create a circle with radius 125
    SimpleCircle circle3 = new SimpleCircle(125);
    System.out.println("The area of the circle of radius "
      + circle3.radius + " is " + circle3.getArea());

    // Modify circle radius
    circle2.radius = 100; // or circle2.setRadius(100)
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());
  }
}
```

● How to Create objects

How to access an object's data and methods

For example : Defining a class named Circle and its three objects.

● **How to access an object's data and methods**

What's the difference between instance variable and the variable defined in the method?

● An instance variable is a variable that is defined in a class, and each object of that class holds a separate copy of that variable.
● On the other hand, a local variables is a variable that can be assessed only within a particular block of code (e.g. function, loop block, etc.) in which it is defined.

```java
public class TestSimpleCircle {
  /** Main method */
  public static void main(String[] args) {
    // Create a circle with radius 1
    SimpleCircle circle1 = new SimpleCircle();
    System.out.println("The area of the circle of radius "
      + circle1.radius + " is " + circle1.getArea());
          Accessing an Object's Data        Accesing an Object's Method
    // Create a circle with radius 25
    SimpleCircle circle2 = new SimpleCircle(25);
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());

    // Create a circle with radius 125
    SimpleCircle circle3 = new SimpleCircle(125);
    System.out.println("The area of the circle of radius "
      + circle3.radius + " is " + circle3.getArea());
                             Modify circle radius
    // Modify circle radius
    circle2.radius = 100; // or circle2.setRadius(100)
    System.out.println("The area of the circle of radius "
      + circle2.radius + " is " + circle2.getArea());
  }
}
```

The difference between private and public

1. **Public:** You can use the public visibility modifier for **classes, methods, and data** fields to denote they **can be accessed from any other classes**.
2. **Default:** If no visibility modifier is used, then by default the **classes, methods, and data** fields are accessible by **any class in the same package**.
3. **Private**: The private modifier makes **methods and data** fields accessible only from **within its own class**.

How to use get and set methods

Why?

To prevent direct modifications of data fields, you should declare **the data fields private**. This is known as **data field encapsulation. A private data field cannot be accessed by an object from outside the class!!**

- To make a private data field accessible, provide **a getter method** to return its value.

```
public returnType getPropertyName()
```

If the returnType is boolean ,

```
public boolean isPropertyName()
```

- To enable a private data field to be updated, provide **a setter method** to set a new value.

```
public void setPropertyName(dataType propertyValue)
```

How to use get and set methods

```java
public class CircleWithPrivateDataFields {
  /** The radius of the circle */
  private double radius = 1;
  /** The number of the objects created */
  private static int numberOfObjects = 0;

  /** Construct a circle with radius 1 */
  public CircleWithPrivateDataFields() {
    numberOfObjects++;
  }
  /** Construct a circle with a specified radius */
  public CircleWithPrivateDataFields(double newRadius) {
    radius = newRadius;
    numberOfObjects++;
  }

  /** Return radius */
  public double getRadius() {
    return radius;
  }
  /** Set a new radius */
  public void setRadius(double newRadius) {
    radius = (newRadius >= 0) ? newRadius : 0;
  }
  /** Return numberOfObjects */
  public static int getNumberOfObjects() {
    return numberOfObjects;
  }

  /** Return the area of this circle */
  public double getArea() {
    return radius * radius * Math.PI;
  }
}
```

```java
public class TestCircleWithPrivateDataFields {
  /** Main method */
  public static void main(String[] args) {
    // Create a Circle with radius 5.0
    CircleWithPrivateDataFields myCircle =
      new CircleWithPrivateDataFields(5.0);
    System.out.println("The area of the circle of radius "
      + myCircle.getRadius() + " is " + myCircle.getArea());

    // Increase myCircle's radius by 10%
    myCircle.setRadius(myCircle.getRadius() * 1.1);
    System.out.println("The area of the circle of radius "
      + myCircle.getRadius() + " is " + myCircle.getArea());
  }
}
```

Using ArrayList in java.util

ArrayList is implemented as a resizable array. As more elements are added to ArrayList, its size is increased dynamically. It's elements can be accessed directly by using the get and set methods, since ArrayList is essentially an array.

java.util.ArrayList<E>	
+ArrayList()	Creates an empty list.
+add(e: E): void	Appends a new element e at the end of this list.
+add(index: int, e: E): void	Adds a new element e at the specified index in this list.
+clear(): void	Removes all elements from this list
+contains(o: Object): boolean	Returns true if this list contains the element o.
+get(index: int): E	Returns the element from this list at the specified index.
+indexOf(o: Object): int	Returns the index of the first matching element in this list.
+isEmpty(): boolean	Returns true if this list contains no elements.
+lastIndexOf(o: Object): int	Returns the index of the last matching element in this list.
+remove(o: Object): boolean	Removes the first element CDT from this list. Returns true if an element is removed.
+size(): int	Returns the number of elements in this list.
+remove(index: int): E	Removes the element at the specified index. Returns the removed element.
+set(index: int, e: E): E	Sets the element at the specified index.

Using ArrayList in java.util

```java
3  import java.util.ArrayList;
4
5  public class TestArrayList {
6      public static void main(String[] args) {
7          // Create a list to store cities          Create an ArrayList
8          ArrayList<String> cityList = new ArrayList<>();
9
10         // Add some cities in the list
11         cityList.add("London");                    Add new elements
12         // cityList now contains [London]
13         cityList.add("Denver");
14         // cityList now contains [London, Denver]
15         cityList.add("Paris");
16         // cityList now contains [London, Denver, Paris]
17         cityList.add("Miami");
18         // cityList now contains [London, Denver, Paris, Miami]
19         cityList.add("Seoul");
20         // contains [London, Denver, Paris, Miami, Seoul]
21         cityList.add("Tokyo");
22         // contains [London, Denver, Paris, Miami, Seoul, Tokyo]
23
24         System.out.println("List size? " + cityList.size());
25         System.out.println("Is Miami in the list? " +
26             cityList.contains("Miami"));
27         System.out.println("The location of Denver in the list? "
28             + cityList.indexOf("Denver"));
29         System.out.println("Is the list empty? " +
30             cityList.isEmpty()); // Print false
31
32         // Insert a new city at index 2
33         cityList.add(2, "Xian");
34         // contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
35
36         // Remove a city from the list
37         cityList.remove("Miami");
38         // contains [London, Denver, Xian, Paris, Seoul, Tokyo]
```

```java
32         // Insert a new city at index 2
33         cityList.add(2, "Xian");
34         // contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]
35
36         // Remove a city from the list
37         cityList.remove("Miami");
38         // contains [London, Denver, Xian, Paris, Seoul, Tokyo]
39
40         // Remove a city at index 1
41         cityList.remove(1);
42         // contains [London, Xian, Paris, Seoul, Tokyo]
43
44         // Display the contents in the list
45         System.out.println(cityList.toString());
46
47         // Display the contents in the list in reverse order
48         for (int i = cityList.size() - 1; i >= 0; i--)
49             System.out.print(cityList.get(i) + " ");
50         System.out.println();
51
52         // Create a list to store two circles
53         java.util.ArrayList<Circle> list
54             = new java.util.ArrayList<>();
55
56         // Add two circles                  Create an ArrayList
57         list.add(new Circle(2));            to store objects
58         list.add(new Circle(3));
59
60         // Display the area of the first circle in the list
61         System.out.println("The area of the circle? " +
62             list.get(0).getArea());
63     }
64 }
```

Exercises

Complete the exercises in the **2020S-Java-A-Lab-7.pdf** and submit to the blackboard as required.

methods of String

```
1
2 public class StringSimpleDemo {
3
4⊙    public static void main(String[] args) {
5        String message = "Welcome to Java";
6        System.out.println("The length of " + message + " is "
7        + message.length());
8
9
10        System.out.printf("The %d character of message is %c\n",0, message.charAt(0));
11
12        // Three strings are concatenated
13        String concat_message = "Welcome " + "to " + "Java";
14        System.out.println(concat_message);
15        String message1 = "Welcome to ";
16        String message2 = "java";
17        System.out.println(message1.concat(message2));
18
19        System.out.println("Welcome".toUpperCase());
20
21    }
22
23 }
```
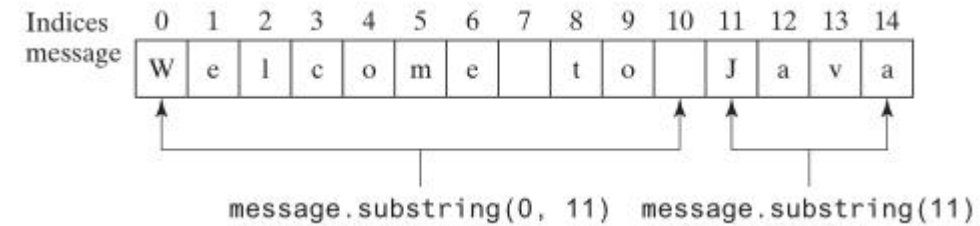
TABLE 4.7 Simple Methods for **String** Objects

Method	Description
length()	Returns the number of characters in this string.
charAt(index)	Returns the character at the specified index from this string.
concat(s1)	Returns a new string that concatenates this string with string s1.
toUpperCase()	Returns a new string with all letters in uppercase.
toLowerCase()	Returns a new string with all letters in lowercase.
trim()	Returns a new string with whitespace characters trimmed on both sides.

TABLE 4.8 Comparison Methods for **String** Objects

Method	Description
equals(s1)	Returns true if this string is equal to string s1.
equalsIgnoreCase(s1)	Returns true if this string is equal to string s1; it is case insensitive.
compareTo(s1)	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than s1.
compareToIgnoreCase(s1)	Same as compareTo except that the comparison is case insensitive.
startsWith(prefix)	Returns true if this string starts with the specified prefix.
endsWith(suffix)	Returns true if this string ends with the specified suffix.
contains(s1)	Returns true if s1 is a substring in this string.

```
1
2 public class StringCampareDemo {
3
4⊙      public static void main(String[] args) {
5
6          String s1 = "Welcome to Java";
7          String s2 = "Welcome to Java";
8          String s3 = "Welcome to C++";
9          System.out.println(s1.equals(s2)); // true
10          System.out.println(s1.equals(s3)); // false
11
12          System.out.println(s1.compareTo(s2)); // 0
13          System.out.println(s1.compareTo(s3)); // >0
14          System.out.println(s3.compareTo(s1)); // <0
15      }
16
17 }
```

methods of String

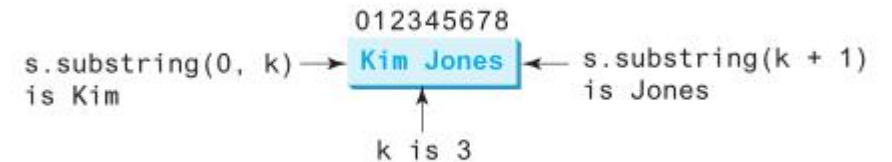**TABLE 4.9** The String Class Contains the Methods for Obtaining Substrings

Method	Description
substring(beginIndex)	Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 4.2.
substring(beginIndex, endIndex)	Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex − 1, as shown in Figure 4.2. Note the character at endIndex is not part of the substring.

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
message	W	e	l	c	o	m	e		t	o		J	a	v	a

message.substring(0, 11)   message.substring(11)

TABLE 4.10 The String Class Contains the Methods for Finding Substrings

Method	Description
index Of (ch)	Returns the index of the first occurrence of ch in the string. Returns −1 if not matched.
indexOf(ch, fromIndex)	Returns the index of the first occurrence of ch after fromIndex in the string. Returns −1 if not matched.
indexOf(s)	Returns the index of the first occurrence of string s in this string. Returns −1 if not matched.
indexOf(s, fromIndex)	Returns the index of the first occurrence of string s in this string after fromIndex. Returns −1 if not matched.
lastIndexOf(ch)	Returns the index of the last occurrence of ch in the string. Returns −1 if not matched.
lastIndexOf(ch, fromIndex)	Returns the index of the last occurrence of ch before fromIndex in this string. Returns −1 if not matched.
lastIndexOf(s)	Returns the index of the last occurrence of string s. Returns −1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of string s before fromIndex. Returns −1 if not matched.

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

```
              012345678
s.substring(0, k) → Kim Jones ← s.substring(k + 1)
is Kim                           is Jones
                    k is 3
```

What's the difference between a character array and a string

Sr. No.	Key	String	Character array
1	Implementation	String refers to a sequence of characters represented as a single data type	Character Array is a sequential collection of data type char
2	Internal implementation	Strings are immutable.	Character Arrays are mutable
3	Built-in Functions	As String is a class so is provided with various built-in function substring(), charAt() etc.	No built in functions are provided in Java for operations on Character Arrays.
4	Concatenation	String can be concatenated either by using + operator or using its built in function concate().	Character array could not use either of these function/operator to get concatenated.
5	Storage	All Strings are stored in the String Constant Pool.	All Character Arrays are stored in the Heap..
6	Conversion	A String can be converted into Character Array by using the toCharArray() method of String class.	On the other hand, a Character Array can be converted into String by passing it into a String Constructor.

What's the difference between a character array and a string

For example:

```java
char [] cs = {'a','b','c'};
String str = cs.toString();
System.out.println("the Length of cs  is : " + cs.Length);
System.out.println("the Length of str is : " + str.Length());
```

```java
String str = "abc";
char [] cs = str.toCharArray();
System.out.println("the Length of cs  is : " + cs.Length);
System.out.println("the Length of str is : " + str.Length());
```

"length" is a attribute of array while "length()" is a method of String

THANK YOU

贾艳红 Jana

Email:jiayh@mail.sustech.edu.cn