Introduction to computer programming A
LAB8

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

LAB OBJECTIVES

- Learn to declare constructors and use them to construct objects
- Learn to declare and use static data fields
- Learn to declare and use the toString( ) method
- Learn to use various String methods

knowledge points

How to define and use constructors

Part 1: Constructors and instance methods

The Circle class defined in the previous lab does not contain any explicitly declared constructor. The Java compiler will provide a default constructor that would initialize all three fields (radius, x, y) to 0.0 when called. If we want to create a circle object, of which the three fields have the following values: radius = 2.0, x = 1.0, y = 1.0, we can write a main method like the one below.

```java
public static void main(String[] args) {
    Circle c = new Circle();
    c.setRadius(2.0);
    c.setX(1.0);
    c.setY(1.0);
}
```

However, this is quite troublesome. A better solution is to declare constructors so that they can be called to construct objects with certain radius values and center positions. The following code declares two such constructors. The first constructor takes one argument to initialize the radius field (the fields x and y will be initialized to 0.0). The second constructor takes three arguments to initialize all three fields.

```java
public Circle(double radius) {
    this.radius = radius;
}

public Circle(double radius, double x, double y) {
    this.radius = radius;
    this.x = x;
    this.y = y;
}
```

How to define and use constructors

Note that in the constructors, "this" keyword is needed to differentiate the field access from method argument access. Now we can simply create the circle (radius = 2.0, x = 1.0, y = 1.0) with a constructor call. Much easier, right?

```java
public static void main(String[] args) {
    Circle c = new Circle(2.0, 1.0, 1.0);
}
```

Continue to type the following code in the main method and see what happens.

```java
Circle c = new Circle();
```

The code would not compile. Do you know why? If not, please go check our lecture notes.

How to define and use constructors

Java compiler distinguish between a __method__ and a constructor by its name and return type.

1. The same name
2. No return value
3. invoked to create an object using the **new** operator

```java
public class ConstructorExample {
                                          3 invoked automatically
    public static void main(String[] args){
        Constructor obj1 = new Constructor();
        System.out.println("Value of x = " + obj1.x);
        Constructor obj2 = new Constructor(10);
        System.out.println("Value of x = " + obj2.x);
    }
}

class Constructor {
    int x;           1 the same name
    // constructor
    Constructor(){   No-Arg Constructor
        System.out.println("No-Arg Constructor Called");
  2 no return value
        x = 5;
    }                Arg Constructor
    Constructor(int x) {
        System.out.println("Arg Constructor Called");
        this.x = x;
    }
}
```

How to define and use constructors

If you do not create constructors yourself, the Java compiler will automatically create a no-argument constructor during run-time. This constructor is known as default constructor. The default constructor initializes any uninitialized instance variables.

Type	Default Value
boolean	false
byte	0
short	0
int	0
long	0L
char	\u0000
float	0.0f
double	0.0d
object	Reference null

DefaultConstructorExample

How to Define and Use Static Variables and Methods

1. Static data field belongs to class instead of object.

```java
public class Circle {
    private double radius;
    private double x;
    private double y;
    private static int cnt = 0;
}
```

cnt is the static data field, it got a inital value 0.
In the class static method could access a static data field.

```java
public static int getCnt(){
    return cnt;
}
```

How about constructor ?

```java
public Circle(double radius) {
    this.radius = radius;
    cnt++;
}
```

Instance data field belongs to object while static data field belongs to class, static data field is shared with all the objects of the class.

How to Define and Use Static Variables and Methods

```java
public class CircleWithStaticMembers {
  /** The radius of the circle */
  private double radius;              // instance variable

  /** The number of the objects created */   // static variable
  private static int numberOfObjects = 0;

  /** Construct a circle with radius 1 */
  CircleWithStaticMembers() {
    radius = 1.0;
    numberOfObjects++;
  }

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

  public double getRadius() {
    return radius;
  }

  public void setRadius(double radius) {
    this.radius = radius;
  }
                                      // static method
  /** Return numberOfObjects */
  static int getNumberOfObjects() {
    return numberOfObjects;
  }
                                      // instance method
  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }
}
```

A static variable is shared by all objects of the class!!

- How to define define instance variables and methods
- How to define and use Static Variables and Methods
- How to create an Object and use instance variables and methods

CircleWithStaticMembers

- radius: double
- numberOfObjects: int
- CircleWithStaticMembers()
- CircleWithStaticMembers(newRadius: double)
- getNumberOfObjects(): int
- getArea(): double
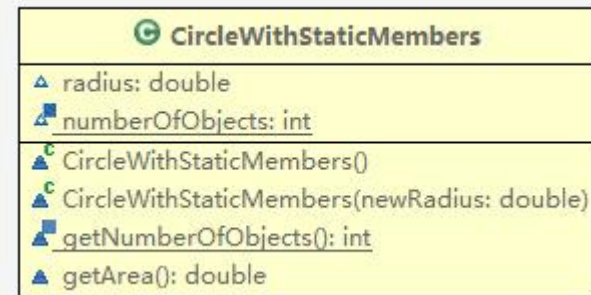
class diagram

How to Define and Use Static Variables and Methods

```java
public class TestCircleWithStaticMembers {
  /** Main method */
  public static void main(String[] args) {
    System.out.println("Before creating objects");
    System.out.println("The number of Circle objects is " +
      CircleWithStaticMembers.getNumberOfObjects());

    // accessing static method

    // Create c1
    CircleWithStaticMembers c1 = new CircleWithStaticMembers();

    // Display c1 BEFORE c2 is created
    System.out.println("\nAfter creating c1");
    System.out.println("c1: radius (" + c1.getRadius() +
      ") and number of Circle objects (" +
      c1.getNumberOfObjects() + ")");

    // Create c2
    CircleWithStaticMembers c2 = new CircleWithStaticMembers(5);

    // Modify c1
    c1.setRadius(9);

    // Display c1 and c2 AFTER c2 was created
    System.out.println("\nAfter creating c2 and modifying c1");
    System.out.println("c1: radius (" + c1.getRadius() +
      ") and number of Circle objects (" +
      c1.getNumberOfObjects() + ")");
    System.out.println("c2: radius (" + c2.getRadius() +
      ") and number of Circle objects (" +
      CircleWithStaticMembers.getNumberOfObjects() + ")");
  }
}
```

● How to use instance variables and methods
● How to use static variables and methods

A static method cannot access instance members (i.e., instance data fields and methods) of the class

The relationship between static and instance members:
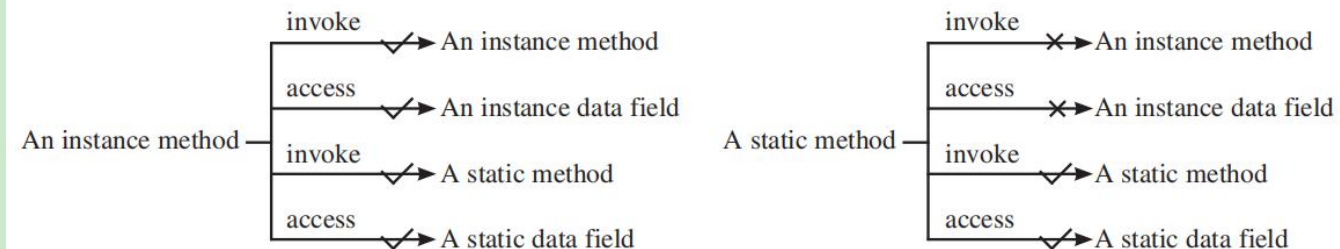
The toString() methods of classes

2. A special instance method **toString( )** which return a string related to the current object.

While the object is used as a string, the toString( ) is invoked by default. For example:

1) Create a Circle object c : `Circle c = new Circle(1.0, 1.0, 1.0);`

2) Use "`System.out.print(c)`" to print the c, what's the output?

3) Use "`System.out.print(c.toString())`" instead of "`System.out.print(c)`", what's the output?

4) Use IDEA to generated the toString() of the current class.
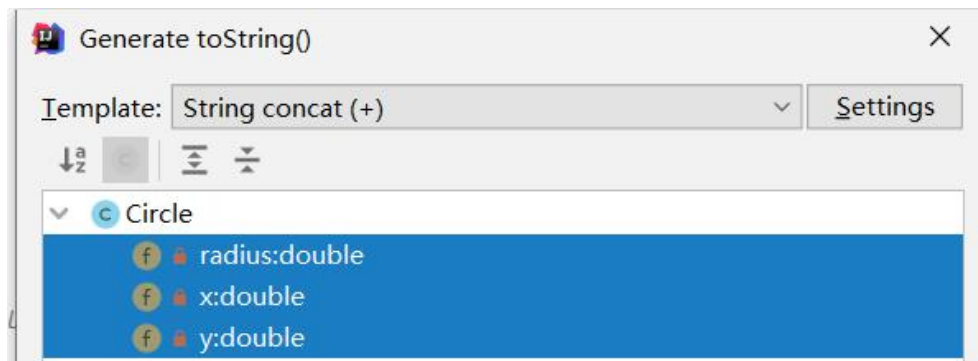
The **toString()** methods of classes

The following instance method toString() is generated, all the instance data field is added into the String:

```java
public String toString() {
    return "Circle{" +
        "radius=" + radius +
        ", x=" + x +
        ", y=" + y +
        '}';
}
```

5) Invoke "System.out.print(c)" and "System.out.print(c.toString())" again, what's the output?

6) While change the toString( ) as follow code, invoke "System.out.print(c)". what's the output, why?

```java
public String toString(char z) {
    return "Circle{" +
        "radius=" + radius +
        ", x=" + x +
        ", y=" + y +
        ", z=" + z +
        '}';
}
```

The **toString()** methods of classes

◆**toString() is always invoked automatically while an object is used as a String**

- such as: x is an integer data, while System.out.println(x) it actrally run as System.out.println(x.toString());
- How about print an object which is a instance of a user defined class, what's the output info?

◆There is a default toString() method which is a part of Class Object, if you want to define a new

- toString(), override it in the your class.

```java
3  public class CircleWithToString {
4      /** The radius of the circle */
5      private double radius = 1;
6
7      /** The number of the objects created */
8      private static int numberOfObjects = 0;
9
10     /** Construct a circle with radius 1 */
11     public CircleWithToString() {
12         numberOfObjects++;
13     }
14
15     /** Construct a circle with a specified radius */
16     public CircleWithToString(double newRadius) {
17         radius = newRadius;
18         numberOfObjects++;
19     }
20
21     public String toString() {
22         return "[radius:" + String.format("%-30f", this.radius) + "area: " + String.format("%-15f", this.getArea())
23                 + "]";
24
25     }
26
```

The toString() methods of classes

```java
/** Print an array of circles and their total area */
public static void printCircleArray(Circle[] circleArray) {
    System.out.printf("%-30s%-15s\n", "Radius", "Area");
    for (int i = 0; i < circleArray.length; i++) {
        System.out.printf("%-30f%-15f\n",
                circleArray[i].getRadius(), circleArray[i].getArea());
    }

    System.out.println("-----------------------------------------------");

    // Compute and display the result
    System.out.printf("%-30s%-15f\n",
            "The total areas of circles is", sum(circleArray));
}
```
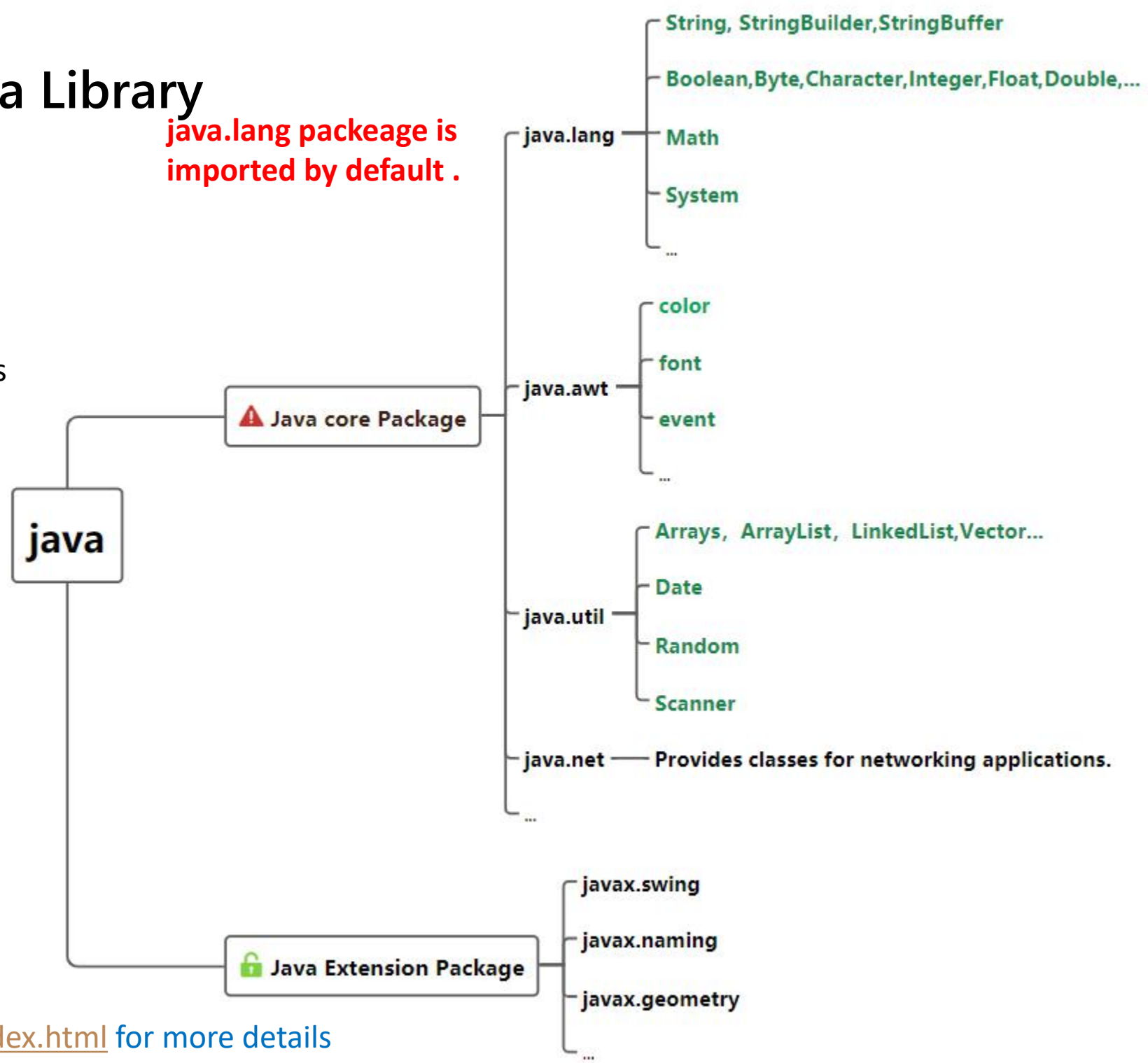
```java
/** Print an array of circles and their total area */
public static void printCircleArray(CircleWithToString[] circleArray) {
    for (int i = 0; i < circleArray.length; i++) {
        System.out.println(circleArray[i]);//or circleArray[i].toString()
    }
}
```

Using Classes from the java Library

java.lang packeage is imported by default .

What is a Java library?

java library, also known as java API. It can help developers develop Java programs quickly and easily. The deployment format of a Java library is a JAR file

```
java
├── ⚠ Java core Package
│   ├── java.lang
│   │   ├── String, StringBuilder, StringBuffer
│   │   ├── Boolean, Byte, Character, Integer, Float, Double,...
│   │   ├── Math
│   │   ├── System
│   │   └── ...
│   ├── java.awt
│   │   ├── color
│   │   ├── font
│   │   ├── event
│   │   └── ...
│   ├── java.util
│   │   ├── Arrays, ArrayList, LinkedList, Vector...
│   │   ├── Date
│   │   ├── Random
│   │   └── Scanner
│   ├── java.net ── Provides classes for networking applications.
│   └── ...
└── 🔒 Java Extension Package
    ├── javax.swing
    ├── javax.naming
    ├── javax.geometry
    └── ...
```

Using Classes from the java Library

A primitive-type value is not an object, but it can be wrapped in an object using a wrapper class in the Java API. Numeric wrapper classes are very similar to each other.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

java.lang.Integer

```
-value: int
+MAX_VALUE: int
+MIN_VALUE: int

+Integer(value: int)
+Integer(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longValue(): long
+floatValue(): float
+doubleValue(): double
+compareTo(o: Integer): int
+toString(): String
+valueOf(s: String): Integer
+valueOf(s: String, radix: int): Integer
+parseInt(s: String): int
+parseInt(s: String, radix: int): int
```

Numeric wrapper classes are very similar to each other

java.lang.Double

```
-value: double
+MAX_VALUE: double
+MIN_VALUE: double

+Double(value: double)
+Double(s: String)
+byteValue(): byte
+shortValue(): short
+intValue(): int
+longValue(): long
+floatValue(): float
+doubleValue(): double
+compareTo(o: Double): int
+toString(): String
+valueOf(s: String): Double
+valueOf(s: String, radix: int): Double
+parseDouble(s: String): double
+parseDouble(s: String, radix: int): double
```

Note

Most wrapper class names for a primitive type are the same as the primitive data type name with the first letter capitalized. The exceptions are **Integer** for **int** and **Character** for **char**.

Using Classes from the java Library

Wrapper class Example:

```java
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1 {
    public static void main(String args[]) {
        //Converting int into Integer
        int a = 20;
        Integer i = Integer.valueOf(a);// converting int into Integer explicitly
        Integer j = a;// autoboxing, now compiler will write Integer.valueOf(a) internally

        System.out.println(a + " " + i + " " + j);


        //Converting Integer to int
        Integer b=new Integer(30);
        int k=b.intValue();//converting Integer to int explicitly
        int n=b;//unboxing, now compiler will write b.intValue() internally

        System.out.println(b+" "+k+" "+n);
    }
}
```

```
20 20 20
30 30 30
```

Using Date and Random in java.util

java.util.Date	
+Date()	Constructs a Date object for the current time.
+Date(elapseTime: long)	Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
+toString(): String	Returns a string representing the date and time.
+getTime(): long	Returns the number of milliseconds since January 1, 1970, GMT.
+setTime(elapseTime: long): void	Sets a new elapse time in the object.

java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (excluding n).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
+nextBoolean(): boolean	Returns a random boolean value.

Using Date and Random in java.util

For Example:

```java
import java.util.*;
public class DateAndRandom {

    public static void main(String[] args) {
        Date date = new Date();
        System.out.println("The elapsed time since Jan 1, 1970 is " +
          date.getTime() + " milliseconds");
        System.out.println(date.toString());


        Random generator1 = new Random(3);
        System.out.print("From generator1: ");
        for (int i = 0; i < 10; i++)
          System.out.print(generator1.nextInt(1000) + " ");
        Random generator2 = new Random(3);
        System.out.print("\nFrom generator2: ");
        for (int i = 0; i < 10; i++)
          System.out.print(generator2.nextInt(1000) + " ");

    }

}
```

```
The elapsed time since Jan 1, 1970 is 1573440941254 milliseconds
Mon Nov 11 10:55:41 CST 2019
From generator1: 734 660 210 581 128 202 549 564 459 961
From generator2: 734 660 210 581 128 202 549 564 459 961
```

String vs StringBuffer vs StringBuilder

Both StringBuilder and StringBuffer can provide similar functionality to Strings,but stores its data in a mutable way.

```java
StringBuilder buf = new StringBuilder();
String [] sx = {"0","1","2","3","4","5"};
    for (String x : sx) {
        buf.append(x);
        buf.append("-");
        System.out.println(buf);
    }
}
```

```
0-
0-1-
0-1-2-
0-1-2-3-
0-1-2-3-4-
0-1-2-3-4-5-
```

String vs StringBuffer vs StringBuilder

Various StringBuilder Methods:

Method	Description
StringBuilder append(boolean b) StringBuilder append(char c) StringBuilder append(char[] str) StringBuilder append(char[] str, int offset, int len) StringBuilder append(double d) StringBuilder append(float f) StringBuilder append(int i) StringBuilder append(long lng) StringBuilder append(Object obj) StringBuilder append(String s)	Appends the argument to this string builder. The data is converted to a string before the append operation takes place.
StringBuilder delete(int start, int end) StringBuilder deleteCharAt(int index)	The first method deletes the subsequence from start to end-1 (inclusive) in the StringBuilder's char sequence. The second method deletes the character located at index.

String vs StringBuffer vs StringBuilder

Method	Description
StringBuilder insert(int offset, boolean b) StringBuilder insert(int offset, char c) StringBuilder insert(int offset, char[] str) StringBuilder insert(int index, char[] str, int offset, int len) StringBuilder insert(int offset, double d) StringBuilder insert(int offset, float f) StringBuilder insert(int offset, int i) StringBuilder insert(int offset, long lng) StringBuilder insert(int offset, Object obj) StringBuilder insert(int offset, String s)	Inserts the second argument into the string builder. The first integer argument indicates the index before which the data is to be inserted. The data is converted to a string before the insert operation takes place.
StringBuilder replace(int start, int end, String s) void setCharAt(int index, char c)	Replaces the specified character(s) in this string builder.
StringBuilder reverse()	Reverses the sequence of characters in this string builder.
String toString()	Returns a string that contains the character sequence in the builder.

String vs StringBuffer vs StringBuilder

An example of reversing a string:

```java
public class ReverseString1 {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
        char[] charArray = new char[len];
        // put original string in an
        // array of chars
        for (int i = 0; i < len; i++) {
            tempCharArray[i] =
                palindrome.charAt(i);
        }
        // reverse array of chars
        for (int j = 0; j < len; j++) {
            charArray[j] =
                tempCharArray[len - 1 - j];
        }
        String reversePalindrome =
            new String(charArray);
        System.out.println(reversePalindrome);
    }
}
```

Output:

```
doT saw I was toD
```

String vs StringBuffer vs StringBuilder

An example of reversing a string:

```java
public class ReverseString2 {

    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        StringBuilder sb = new StringBuilder(palindrome);
        sb.reverse();   // reverse it
        System.out.println(sb);
    }

}
```

Note that println() prints a string builder, because sb.toString() is called implicitly,

Output:

```
doT saw I was toD
```

String vs StringBuffer vs StringBuilder

StringBuffer vs StringBuilder

StringBuffer
1. Thread-Safe
2. Synchronized
3. Since Java 1.0
4. Slower

StringBuilder
1. Not Thread-Safe
2. Not Synchronized
3. Since Java 1.5
4. Faster

As **StringBuilder** isn't thread safe you can't use it in more than one thread.
For a multi-thread environment, use **StringBuffer** instead which does the same and is thread safe

Exercises

Complete the exercises in the **2020S-Java-A-Lab-8.pdf** and submit to the blackboard as required.

THANK YOU

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