# Some Standard Class

Wednesday, December 21, 2022 6:00 PM

Last time, we saw class like String. Recall the three most frequently-used method of String. Length(), indexof(), substring(). Today, we will see class like Integer, Double, and Math

`int length()`

Returns the length of this string.

`String substring(int startIndex)`

Returns a new string that is a substring of this string. The substring starts with the character at `startIndex` and extends to the end of the string. The first character is at index zero. The method throws an `IndexOutOfBoundsException` if `startIndex` is negative or larger than the length of the string. Note that if you're using Java 7 or above, you will see the error `StringIndexOutOfBoundsException`. However, the AP Java subset lists only `IndexOutOfBoundsException`, which is what they will use on the AP exam.

In this case, the start index cannot be larger than the length of string

`String substring(int startIndex, int endIndex)`

Returns a new string that is a substring of this string. The substring starts at index `startIndex` and extends to the character at `endIndex-1`. (Think of it this way: `startIndex` is the first character that you want; `endIndex` is the first character that you *don't* want.) The method throws a `StringIndexOutOfBoundsException` if `startIndex` is negative, or `endIndex` is larger than the length of the string, or `startIndex` is larger than `endIndex`.

In this case, the end index cannot be larger than the length of string

`int indexOf(String str)`

Returns the index of the first occurrence of `str` within this string. If `str` is not a substring of this string, `-1` is returned. The method throws a `NullPointerException` if `str` is null.

Here are some examples:

注意这些都是instance method，应该用个string instance来call

```
"unhappy".substring(2)      //returns "happy"
"cold".substring(4)         //returns "" (empty string)
"cold".substring(5)         //StringIndexOutOfBoundsException
"strawberry".substring(5,7)  //returns "be"
"crayfish".substring(4,8)   //returns "fish"
"crayfish".substring(4,9)   //StringIndexOutOfBoundsException
"crayfish".substring(5,4)   //StringIndexOutOfBoundsException

String s = "funnyfarm";
int x = s.indexOf("farm");  //x has value 5
x = s.indexOf("farmer");    //x has value -1
int y = s.length();         //y has value 9
```

One more thing to remember is that when you are comparing object, you cannot use "==". "==" only compare object reference. If you want to compare content of object like string, you should use **equals**.

## Wrapper class: Integer

Basically, Integer class is used to make primitive type int to an object.

## The Integer Class

The Integer class wraps a value of type int in an object. An object of type Integer contains just one instance variable whose type is int.

Here are the Integer methods you should know for the AP exam:

`Integer(int value)`

Constructs an Integer object from an int. (Boxing.)

`int compareTo(Integer other)`

Returns 0 if the value of this Integer is equal to the value of other, a negative integer if it is less than the value of other, and a positive integer if it is greater than the value of other.

`int intValue()`

Returns the value of this Integer as an int. (Unboxing.)

`boolean equals(Object obj)`

Returns true if and only if this Integer has the same int value as obj.

NOTE

1. This method overrides equals in class Object.
2. This method throws a ClassCastException if obj is not an Integer.

`String toString()`

Returns a String representing the value of this Integer.

Here are some examples to illustrate the Integer methods:

```
Integer intObj = new Integer(6);  //boxes 6 in Integer object
int j = intObj.intValue();        //unboxes 6 from Integer object

System.out.println("Integer value is " + intObj);
//calls toString() for intObj
//output is
//Integer value is 6
```

```
Integer intObj2 = new Integer(3);
int k = intObj2.intValue();
if (intObj.equals(intObj2))       //OK, evaluates to false
    ...
if (intObj.intValue() == intObj2.intValue())
    ...              //OK, since comparing primitive types
```

## The Double Class

The Double class wraps a value of type double in an object. An object of type Double contains just one instance variable whose type is double.

The methods you should know for the AP exam are analogous to those for type Integer.

`Double(double value)`

Constructs a Double object from a double. (Boxing.)

`double doubleValue()`

Returns the value of this Double as a double. (Unboxing.)

`int compareTo(Double other)`

Returns 0 if the value of this Double is equal to the value of other, a negative integer if it is less than the value of other, and a positive integer if it is greater than the value of other.

`boolean equals(Object obj)`

This method overrides equals in class Object and throws a ClassCastException if obj is not a Double. Otherwise it returns true if and only if this Double has the same double value as obj.

`String toString()`

Returns a String representing the value of this Double.

**Let's go to the most important topic of today's lecture, the Math Class.**

## THE Math CLASS

This class implements standard mathematical functions such as absolute value, square root, trigonometric functions, the log function, the power function, and so on. It also contains mathematical constants such as $\pi$ and $e$.

Here are the functions you should know for the AP exam:

`static int abs(int x)`

Returns the absolute value of integer $x$.

`static double abs(double x)`

Returns the absolute value of real number $x$.

`static double pow(double base, double exp)`

Returns $\texttt{base}^{\texttt{exp}}$. Assumes `base` $> 0$, or `base` $= 0$ and `exp` $> 0$, or `base` $< 0$ and `exp` is an integer.

`static double sqrt(double x)`

Returns $\sqrt{x}$, $x \geq 0$.

`static double random()`

Returns a random number $r$, where $0.0 \leq r < 1.0$. (See the next section, Random Numbers.)

All of the functions and constants are implemented as static methods and variables, which means that there are no instances of `Math` objects. The methods are invoked using the class name, `Math`, followed by the dot operator.

Remember to import Math before using it

```java
import java.lang.Math;

public class Lecture6 {
    Run | Debug
    public static void main (String args[]) {

        double x = -2.4;

        System.out.println(Math.abs(x)); // the result is 2.4

        System.out.println(Math.pow(a: 2, b: 2)); // the result is 4

        System.out.println(Math.pow(a: 2, -2)); // the result is 0.25

        System.out.println(Math.pow(-2, b: 2)); // the result is 4

        System.out.println(Math.pow(-2, -2)); // the result is 0.25

        System.out.println(Math.pow(a: 2, b: 2.5)); // the result is 5.6569

        System.out.println(Math.sqrt(a: 2)); // the result is 1.41


    }
}
```

**The most important method: random**

`static double random()`

Returns a random number $r$, where $0.0 \leq r < 1.0$. (See the next section, Random Numbers.)

**Example 1**

Produce a random real value $x$ in the range $0.0 \le x < 6.0$.

```
double x = 6 * Math.random();
```

**Example 2**

Produce a random real value $x$ in the range $2.0 \le x < 3.0$.

```
double x = Math.random() + 2;
```

**Example 3**

Produce a random real value $x$ in the range $4.0 \le x < 6.0$.

```
double x = 2 * Math.random() + 4;
```

In general, to produce a random real value in the range `lowValue` $\le x <$ `highValue`:

```
double x = (highValue - lowValue) * Math.random() + lowValue;
```

## RANDOM INTEGERS

Using a cast to `int`, a scaling factor, and a shifting value, `Math.random()` can be used to produce random integers in any range.

**Example 1**

Produce a random integer, from 0 to 99.

```
int num = (int) (Math.random() * 100);
```

In general, the expression

```
(int) (Math.random() * k)
```

produces a random `int` in the range $0, 1, \ldots, k-1$, where $k$ is called the scaling factor. Note that the cast to `int` truncates the real number `Math.random() * k`.

**Example 2**

Produce a random integer, from 1 to 100.

```
int num = (int) (Math.random() * 100) + 1;
```

In general, if $k$ is a scaling factor, and $p$ is a shifting value, the statement

```
int n = (int) (Math.random() * k) + p;
```

produces a random integer $n$ in the range $p, p+1, \ldots, p+(k-1)$.

**Example 3**

Produce a random integer from 5 to 24.

```
int num = (int) (Math.random() * 20) + 5;
```

Note that there are 20 possible integers from 5 to 24, inclusive.

Practice examples:

1. Here is a program segment to find the quantity $\texttt{base}^{\texttt{exp}}$. Both `base` and `exp` are entered at the keyboard.

```
System.out.println("Enter base and exponent: ");
double base = IO.readDouble();   //read user input
double exp = IO.readDouble();    //read user input
/* code to find power, which equals base^exp */
System.out.print(base + " raised to the power " + exp);
System.out.println(" equals " + power);
```

Which is a correct replacement for
*/* code to find* `power`, *which equals* $\texttt{base}^{\texttt{exp}}$ **/*?

I
```
double power;
Math m = new Math();
power = m.pow(base, exp);
```

II
```
double power;
power = Math.pow(base, exp);
```

III
```
int power;
power = Math.pow(base, exp);
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

2. Consider the squareRoot method defined below:

```
/** @param d a real number such that d >= 0
 *  Postcondition: Returns a Double whose value is the square
 *                 root of the value represented by d.
 */
public Double squareRoot(Double d)
{
    /* implementation code */
}
```

Which `/* implementation code */` satisfies the postcondition?

I
```
double x = d.doubleValue();
x = Math.sqrt(x);
return new Double(x);
```

II
```
return new Double(Math.sqrt(d.doubleValue()));
```

III
```
return (Double) Math.sqrt(d.doubleValue());
```

(A) I only
(B) I and II only
(C) I and III only
(D) II and III only
(E) I, II, and III

3. Here are some examples of negative numbers rounded to the nearest integer.

| Negative real number | Rounded to nearest integer |
|---|---|
| −3.5 | −4 |
| −8.97 | −9 |
| −5.0 | −5 |
| −2.487 | −2 |
| −0.2 | 0 |

Refer to the declaration

```
double d = -4.67;
```

Which of the following correctly rounds `d` to the nearest integer?

(A) `int rounded = Math.abs(d);`

(B) `int rounded = (int) (Math.random() * d);`

(C) `int rounded = (int) (d - 0.5);`

(D) `int rounded = (int) (d + 0.5);`

(E) `int rounded = Math.abs((int) (d - 0.5));`

6. Consider the code segment

```
Integer i = new Integer(20);
/* more code */
```

Which of the following replacements for `/* more code */` correctly sets `i` to have an integer value of 25?

I
```
i = new Integer(25);
```

II
```
i.intValue() = 25;
```

III
```
Integer j = new Integer(25);
i = j;
```

(A) I only
(B) II only
(C) III only
(D) I and III only
(E) II and III only

**Homework preview**

```java
import java.lang.Math;

public class homework5 {
    Run | Debug
    public static void main (String args[]) {

        // Problem 2
        // throw a die for 10000 times, and count how many 6 appears. Print it out.

        // Problem 3
        // throw a die for 10000 times, and count what is the percentage of time that your number is greater than 4. Print it out.

        // Problem 4
        // Suppose that you throw a die. What is the probability that your number is an odd number? (solve it by simulation)
        // You can throw a die for 10000 times and report the percentage of time that your number is odd.

        // Problem 5
        // Suppose you throw two dies and record their numbers. What is the probability that the sum of these two numbers is equal to 10?

        // Problem 7
        // What is the average waiting time until next fish? (solve it by simulation)
        // you can fish for 10000 times, and calculate the average waiting time of these 10000 fishes.

    }

    // Problem 1
    // suppose you want to throw a die and record the number (generate a random integer between 1-6)
    public static int Throw_a_die() {
        return 0;
    }

    // Problem 6
    // suppose you are fishing, and the time until next fish comes is between 5-30 mins uniformly.
    // you want to simulate the time until next fish comes. (generate a random number between 5-30)
    public static double fish_time() {
        return 0.0;
    }


}
```