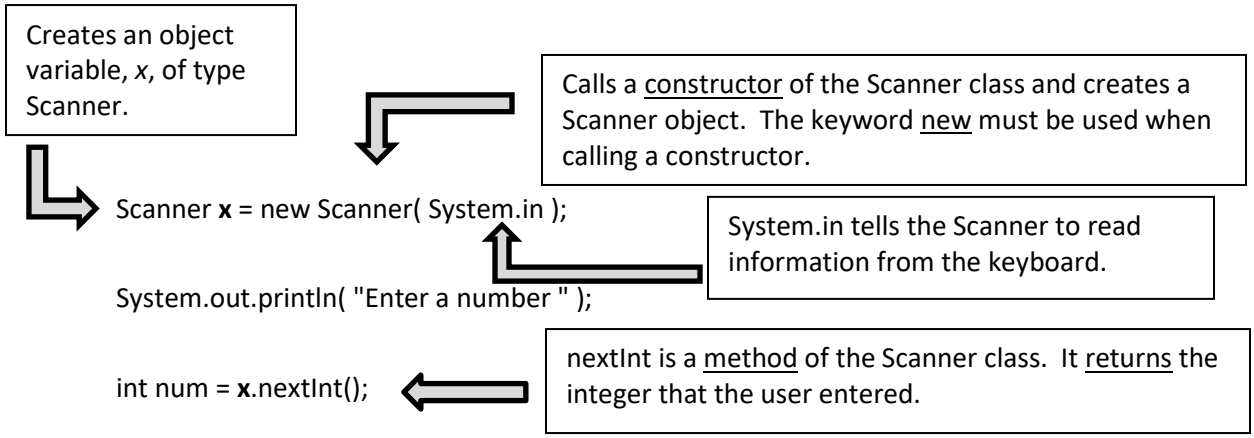
AP CS A
Unit 2. Using Objects. Notes

Classes and Objects. A class is generally used to serve as a blueprint to create objects. An object typically contains variables and has methods that allow it to do something.



Once the object is created (aka _____), you can use its methods. To call a method you need to know:

- Its name.
- What parameters, if any, it has. These are the inputs to a method.
- What the method returns (aka its return type)

Let's look at some methods of the Scanner class.

Return Type	Scanner method name	Parameters
	nextInt	

Consider the following code:

```
Dog fido = new Dog( "Timmy" );          // calls a Dog constructor
int a = fido.fetch( 14.8 );             // calls a method
double b = fido.good( true, 100 );      // calls a different method
```

Return Type	Dog method name	Parameters

Do exercises 1 to 10.

To help illustrate the concepts of classes, objects, and methods. We will work with a simple rectangle class that has only a few methods.

A **Rectangle** object represents a rectangle on a coordinate plane. The sides are parallel to the x and y axes. The vertices are restricted to integer values. This *NOT* the same Rectangle class as the one defined in the Java library of classes.

Here are the constructors of our Rectangle class

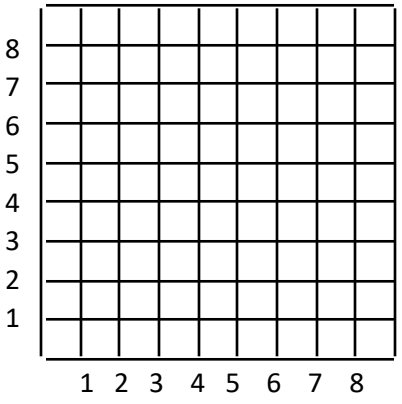
Name(parameters)	**Comments**
Rectangle(int x1, int y1, int x2, int y2)	(x1, y1) is the lower left-hand corner. (x2, y2) is the upper right-hand corner.
Rectangle(int width, int height)	The lower left-hand corner is at the origin. width and height are the dimensions of the rectangle

Methods of our Rectangle class

Return Type	**Name(parameters)**	**Comments**
int	getX()	Returns the x coordinate of the lower left-hand corner
int	getY()	Returns the y coordinate of the lower left-hand corner
boolean	contains(int x, int y)	Returns true if (x,y) is a point within the rectangle. Returns false if it is on the edge or outside the rectangle.
int	area()	Returns the rectangle's area
void	translate(int dx, int dy)	Adds dx to the x coordinate and dy to the y coordinate. Does not change the rectangle's size.
double	diagonal()	Returns the length of the rectangle's diagonal

_____ means that a method returns nothing.

```
Rectangle r1 = new Rectangle( 5, 6, 7, 8 );
Rectangle r2 = new Rectangle( 4, 3 );
boolean b = r2.contains( 1, 2 );
System.out.println( b );          // _________________
int n = r1.getY();
System.out.println( n );          // _________________
r2.translate( 0, 1 );
int a = r2.area();
System.out.println( a );          // _________________
double len = r2.diagonal();
System.out.println( len );        // _________________
```



How many Rectangle objects are instantiated in the above code? _____

Do exercises 11 to 14.

The String Class. A String object represents a sequence of one or more characters where a character could be a letter, digit, or punctuation mark. Each character in a string has a unique index starting at _____. Anything in quotes is a _____ and is an object of the String class.

"jump now"; // the *u* is at index _____, the *n* is at index _____

Classes have _____ that are used to create objects.

String a = new String("easy 123"); _____

However, the String constructor does not have to be explicitly called, you can do this:

_____

The **concatenation** operator is _____. It is used to join a string to another string or a primitive type such a _____.

The addition and concatenation operators have the same level of precedence and are evaluated _____

For example:

```
String a = "MY";
a = a + a;
System.out.println(a);          // _____

String b = "40";
b = b + 3;
System.out.println(b);          // _____

String c = "a" + 3 + 4;
System.out.println(c);          // _____

String d = 3 + 4 + "a";
System.out.println(d);          // _____
```

Note. While you can concatenate a string with an int or double or boolean, you cannot assign these values to a string.

int a = 23;
int b = 1;
String c = a + b;
	int a = 23;
	int b = 1;
	String c = a + b + "";

The last line	The last line

_____ sequences start with a \ and have a special meaning in Java. You are responsible for knowing three of these sequences: \", \n, and \\.

Since quotes (") mark the beginning and end of a string, we must use an escape sequence if we want the include quotes as part of a string.

```
String s = "He said \"GOAT\"?";
System.out.println( s );          // _____
```

If we want a line break to be part of a string, then we must use a different escape sequence.

```
String s = "One\nTwo";
System.out.println( s );          // _____
```

Since \ is used to start an escape sequence, if we want to print a \ then we use an escape sequence.

```
String s = "\\/";
System.out.println(s);            // _____
```

Do exercises 15 to 24.

String Methods.

While the String class has many methods, we only cover a few of them.

The **length** method returns the number of characters in a string.

```
String s = "A cat";          // there is one space between the two words
int n = s.length();          // _____
```

The **substring** method returns a portion of the string. This method is _____

which means that there are multiple methods with the same name but

_____.

```
String a = "ABCDEF";
String b = a.substring( 1, 3 );      // Returns the string in the range [ 1, 3 )
System.out.println( b );             // _____
String c = a.substring( 4 );         // Returns the string in the range [4, to the end]
System.out.println( c );             // _____
```

IMPORTANT. The substring method does NOT _____.
It returns a new string.

Do exercises 25 to 34.

The **indexOf** method searches a string for another string. This method is also overloaded. Here is one version.

```
String a = "there were others";
int n = a.indexOf( "er" );                  // return the index of the first occurrence of er
System.out.println( n );                    // _____________________
n = "haystack".indexOf("needle");           // string literals are string objects
System.out.println( n );                    // returns _____________ if not found
```

Another version of indexOf has two parameters: the string that is looked for and the index where the search should start. The search always goes to the right.

```
String a = "there were others";
int n = a.indexOf( "er", 2 );
System.out.println( n );                    // _____________________
n = a.indexOf( "er", 3 );
System.out.println( n );                    // _____________________
n = "bobble".indexOf( "bob", 2);
System.out.println( n );                    // _____________________
```

NOTE. indexOf is case-sensitive.

```
String a = "Every one";
int k = a.indexOf("e");
System.out.println( k );                    // _____________________
```

The **equals** method returns true if two strings are equal; otherwise it returns false. This is case-sensitive.

```
String a = "cat";
String b = "CAT";
boolean c = a.equals( b );
System.out.println( c );
```

The **toLowerCase** method returns a copy of the string where all the letters are lower-case.

```
String a = "The 4 STARS!";
String b = a.toLowerCase();
System.out.println( b );            // the 4 star!
```

The toLowerCase method is often used in combination with other methods that are case-sensitive. For example, we might first convert two strings to lower-case letters before checking if they are equal or before calling the indexOf method.

The **compareTo** method has one parameter, a string, and returns an integer. The integer is positive if the string is greater than the parameter, negative if the string is less than the parameter, or zero the string equals the parameter.

What does it mean for one string to be greater than or less than another string? In the simple case (which is the only situation we will consider), if two strings are both all lower-case or all upper-case, then here are the rules:

- Letters that come earlier in alphabet are _____ than letters that come later.
- If the first two letters in a string are the same, then compare the next two letters, and so on.
- If both strings are identical except one string has additional letters at the end (e.g. “ax” and “axes” then the shorter string is _____ than the longer string.

Another way to say this is, words that come earlier in the dictionary are _____ than words that come later in the dictionary.

Remember, this assumes that both strings consist of only upper-case letters or only lower-case letters. We will not use the compareTo method with any kinds of other strings.

1. This prints
a) a negative number then a positive number.
b) a positive number then a negative number.
c) two positive numbers.
d) two negative numbers.
	String a = "ADAM";
	String b = "BETTY";
	int n1 = a.compareTo(b);
	System.out.println(n1);
	n1 = b.compareTo(a);
	System.out.println(n1);

2. This prints
a) a negative number then a positive number.
b) a positive number then a negative number.
c) two positive numbers.
d) two negative numbers.
	String a = "milk";
	String b = "and cookies";
	int n1 = a.compareTo(b);
	System.out.println(n1);
	n1 = b.compareTo(a);
	System.out.println(n1);

3. This prints
a) a negative number then a positive number.
b) a positive number then a negative number.
c) two positive numbers.
d) two negative numbers.
	String a = "window";
	String b = "win";
	int n1 = a.compareTo(b);
	System.out.println(n1);
	n1 = b.compareTo(a);
	System.out.println(n1);

Do exercises 35 to 47.

The **Math Class**. In general, methods are associated with objects because they return information about the object or do something to the object. For instance, consider the String class's length method. It returns the number of characters in a specific String object. You must have a specific String to use its length method.

However, there are methods that are associated with a class, and not the objects of that class. These methods are called _____ methods. To call them, you refer to their class, not an object.

The Math class has many static methods; here are five that you need to know.

```
double n = Math.sqrt( 16 );          // returns the _____________________________
n = Math.sqrt( n );
System.out.println( n );             // _____________________________
```

Notice how the *sqrt* method is called. We do NOT create a Math object; we simply use the name of the class.

The return type of the *sqrt* method is _____. If need be, we can cast the returned value to an int.

```
int n = (int) Math.sqrt( 81 );
System.out.println( n );             // ___________________
n = (int) Math.sqrt( 80 );
System.out.println( n );             // ___________________
```

Note: calling a method has higher precedence than the casting operator. The method is called first and then the returned value is cast to an int before the value is assigned to n.

****

The *abs* method returns the absolute value of the parameter.

```
int d = Math.abs( -7 );
System.out.println( d );             // ___________________
double e = Math.abs( -12.7 );
System.out.println( e );             // ___________________
```

In the above example, the first time we called the *abs* method, it returned an int and the second time it returned a double. It can do that because this method is _____.

If the parameter is an _____, then it returns an _____.

If the parameter is a _____, then it returns a _____.

****

The *pow* method has two parameters, both doubles, and returns a double. It returns the value of the first parameter raised to the second parameter.

```
double num = Math.pow( 3, 4 );          // _____
System.out.println( num );              // _____
```

The *random* method returns a random number in the range [0, 1).

```
double a = Math.random();
System.out.println( a );                // 0.5673112178583647
a = Math.random();
System.out.println( a );                // 0.06784309086038653
```

If you ran this code again, you would get two different numbers. Without getting into details, Math.random returns pseudorandom decimals: numbers that appear to be random but are actually based on an algorithm and the time when the code was executed.

While these random decimals can be useful, in many of the programs that you will write, you need to generate a random integer within some range. Here is a formula to generate a random integer in the range [min, max] with an equal likelihood of any of the numbers being returned.

```
int n = (int) ( range * Math.random() ) + min;     // where range = max – min + 1
```

For example, to generate a random integer in the range [-1, 3]

```
int n = (int) ( _________ * Math.random() ) _____________;       // num will be:  -1, 0, 1, 2, or 3
```

1. *a* is a random integer in the range: [_____, _____]	int a = (int)(5 * Math.random()) + 3;
2. *b* is a random integer in the range: [_____, _____]	int b = (int)(12 * Math.random());
3. *c* is a random integer in the range: [_____, _____]	int c = (int)(3 * Math.random()) - 7;

You need to memorize the formula for generating a range of random integers.

Do exercises 48 to the end.