JAVA BASICS

Road Runners Training

CLASE I

Globant
we are ready

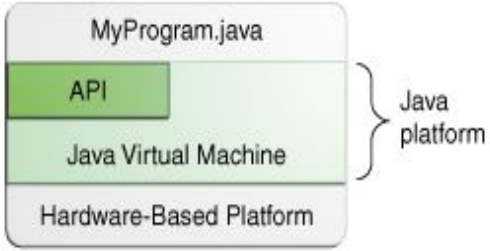
What is JAVA?

Java is a **programming language**

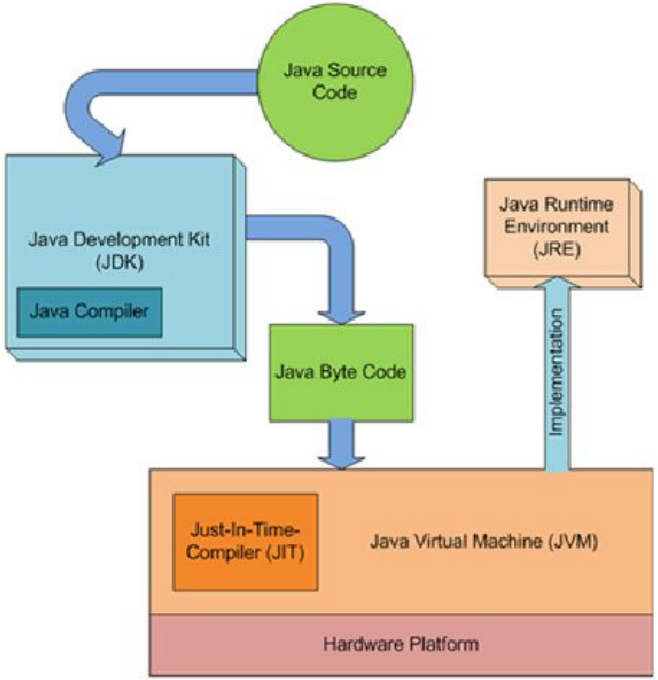
- ✔ **High Level:** refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with variables, arrays, objects.
- ✔ **Robust:** Java is Robust because it is highly supported language. It is portable across many Operating systems. Java also has feature of Automatic memory management and garbage collection. There is a lack of pointers that avoids security problems. There are exception handling and type checking mechanisms.
- ✔ **Secured:** In Java, you cannot access out-of-bound arrays, and you don't have pointers, and thus several security flaws like stack corruption or buffer overflow are not possible to exploit in Java.
- ✔ **Object-oriented**: Java is an OO programming language, that means that it supports Objects, Classes, Inheritance, Polymorphism and Encapsulation.

Java **is also a platform**

A platform is a software environment in which a program runs.

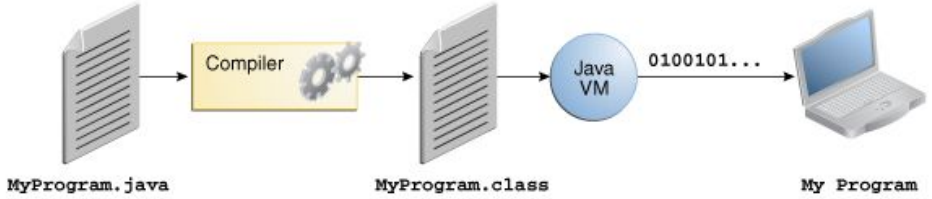


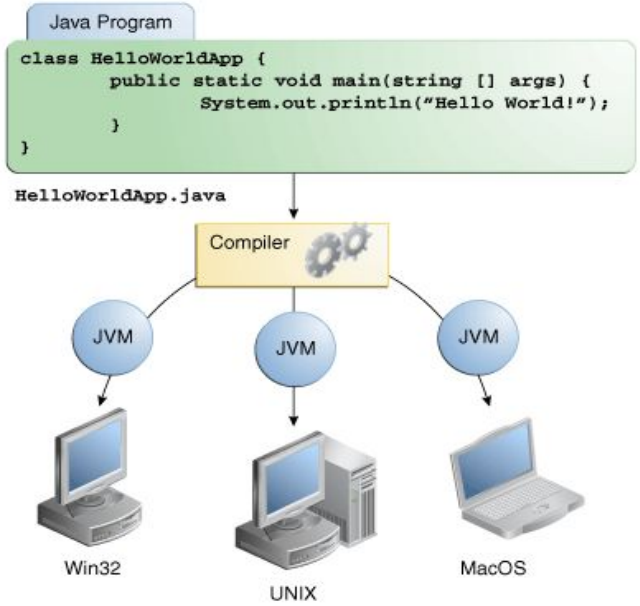
JDK, JVM, JRE and JIT



- **Java Virtual Machine (JVM)** is an abstract computing machine.
- **Java Runtime Environment (JRE)** is an implementation of the JVM.
- **Java Development Kit (JDK)** contains JRE along with various development tools like Java libraries, Java source compilers, Java debuggers, bundling and deployment tools.
- **Just In Time compiler (JIT)** is runs after the program has started executing, on the fly. It has access to runtime information and makes **optimizations of the code** for better performance.

Our first JAVA Application





Object Oriented Programming

Methodology or paradigm to design a program using classes and objects.
It simplifies the software development and maintenance by providing some concepts:

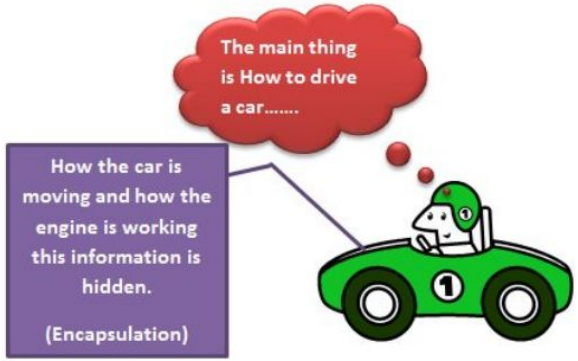
- ✔ Object
- ✔ Class
- ✔ Inheritance
- ✔ Polymorphism
- ✔ Encapsulation

Classes and Objects

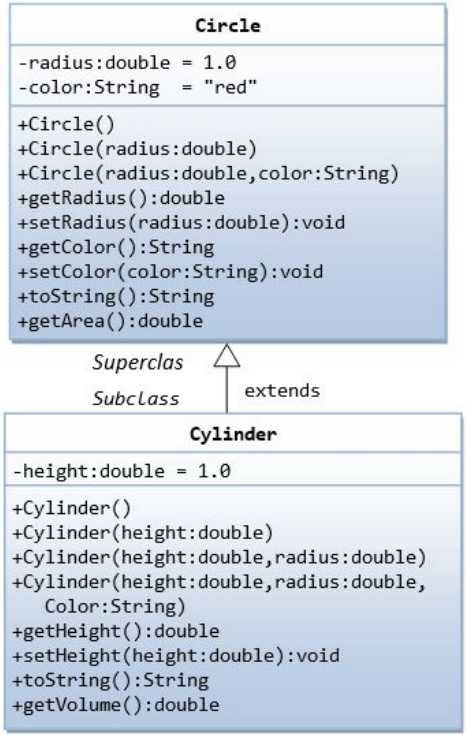
Classes group variables and operations together in coherent modules.
A class can have fields, constructors and methods.
Objects are instances of classes. When you create an object, that object is of a certain class.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here



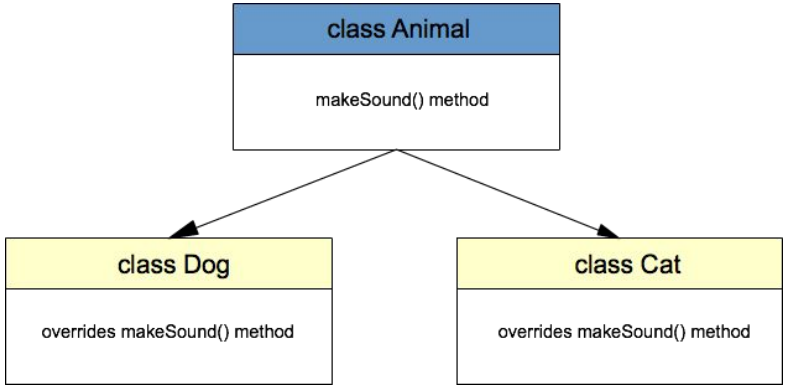
Inheritance



Cylinder.java

```java
/*
 * A Cylinder is a Circle plus a height.
 */
public class Cylinder extends Circle {
   // private instance variable
   private double height;

   // Constructors
   public Cylinder() {
      super();  // invoke superclass' constructor Circle()
      this.height = 1.0;
   }
   public Cylinder(double height) {
      super();  // invoke superclass' constructor Circle()
      this.height = height;
   }
   public Cylinder(double height, double radius) {
      super(radius);  // invoke superclass' constructor Circle(radius)
      this.height = height;
   }
   public Cylinder(double height, double radius, String color) {
      super(radius, color);  // invoke superclass' constructor Circle(radius, color)
      this.height = height;
   }
```

Polymorphism




```java
package net.javatutorial;

public class Dog extends Animal{

        @Override
        public void makeSound() {
                System.out.println("the dog barks");
        }

}
```

```java
package net.javatutorial;

public class Animal {

        public void makeSound() {
                System.out.println("the animal makes sounds");
        }

}
```

```java
package net.javatutorial;

public class PolymorphismExample {

        public static void main(String[] args) {
                Animal animal = new Animal();
                animal.makeSound();
                Dog dog = new Dog();
                dog.makeSound();
                animal = new Cat();
                animal.makeSound();
        }

}
```

Object Class

Every class in the Java system is a descendant (direct or indirect) of the Object class

- **boolean equals(Object obj) :**

This method indicates whether some other object is "equal to" this one

“Example”.equals(“Example”) >> true

- **Class<?> getClass():**

This method returns the runtime class of this Object

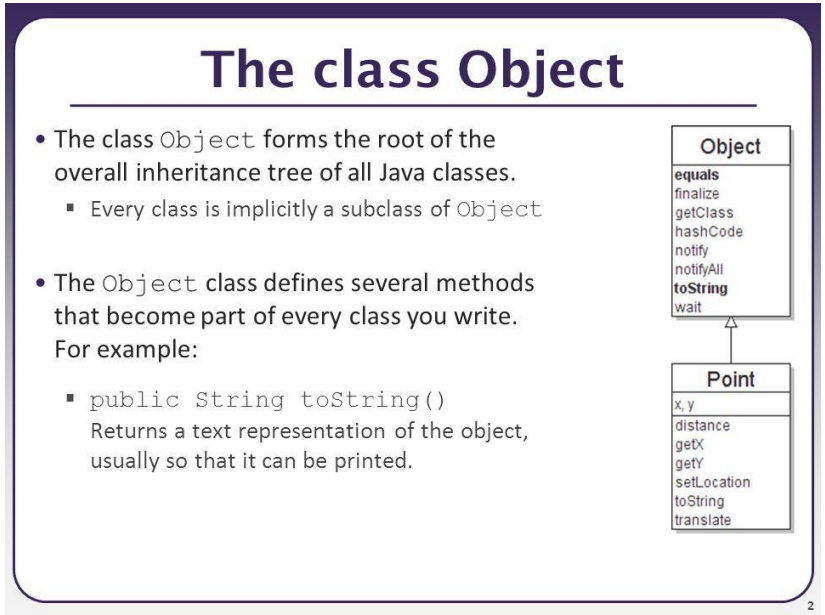
- **Person person = new Person();**

person.getClass() >> class Person

- **String toString():**

This method returns a string representation of the object

“Example”.toString() >> Example



Java Packages

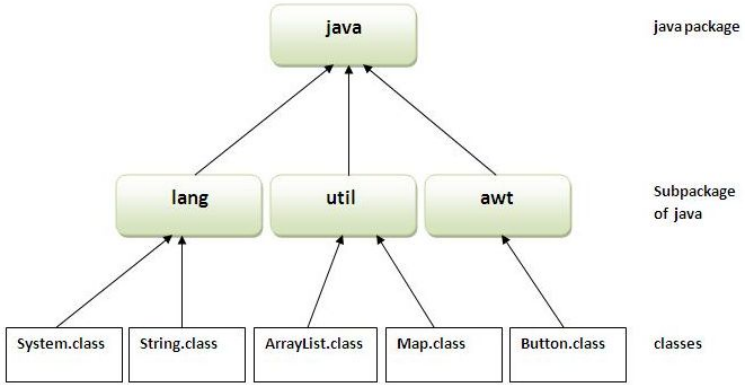
A package is a group of similar types of classes, interfaces and sub-packages.
In Java they can be categorized in:

- ✔ Built-in package.
- ✔ User-defined package.

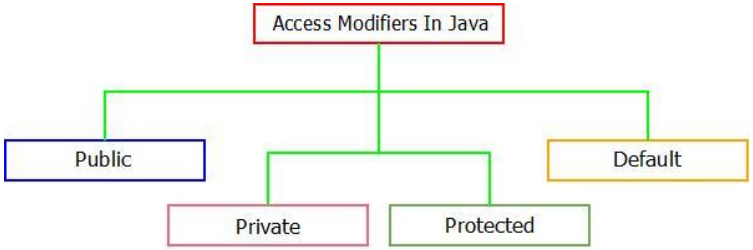
There are many built-in packages such as java, lang, awt, javax, etc.

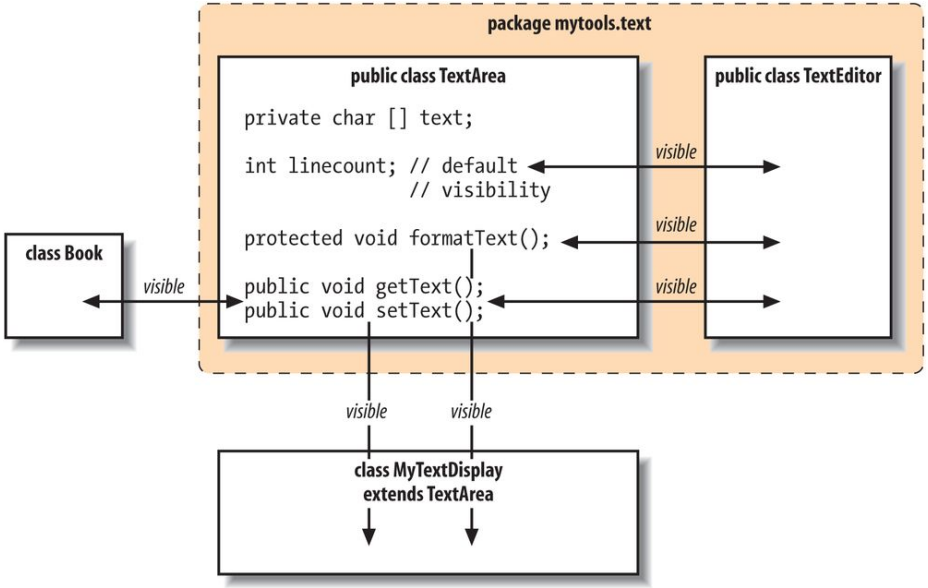
Advantages

1) Used to categorize the classes and interfaces so that they can be easily maintained.
2) Provides access protection.
3) There is no problem with naming collision.



Access Modifiers





- ✔ **Public**: Accessible from outside the package
- ✔ **Protected**: Accessible to classes and interfaces in its own package and derived classes in other packages
- ✔ **Default**: If no access modifier is defined, accessible only inside the package
- ✔ **Private**: Accessible only inside the class where it is defined

JAVA BASICS

Road Runners Training

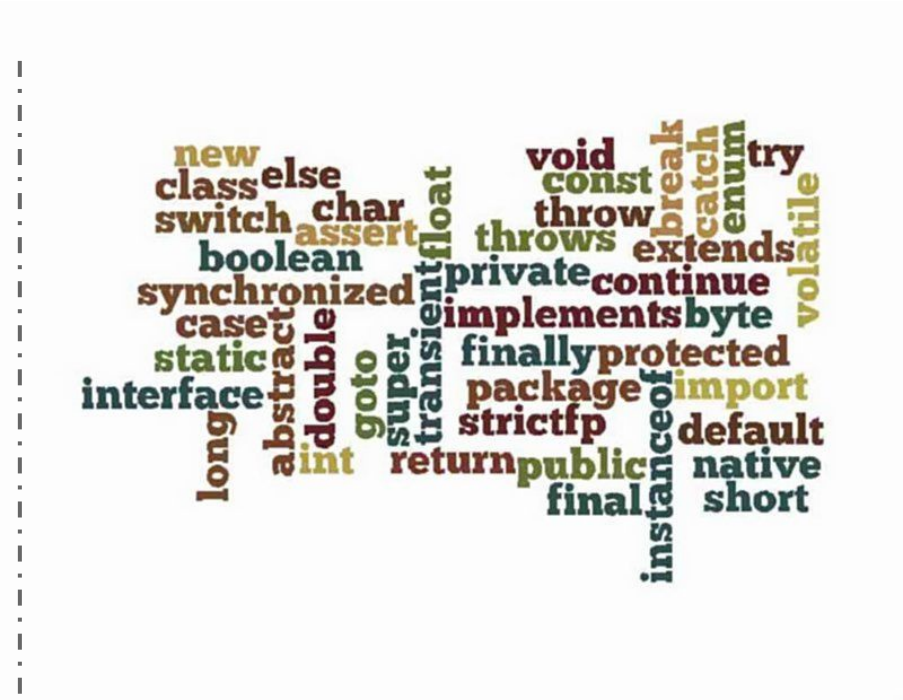
CLASE II

Globant
we are ready

Java Basic Syntax

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

Reserved words



Main Method

The method should not return a value; its return type must be `void`.

The name of the method must be `main`.

The method must accept a `String` array. The name of the method parameter can be any valid identifier name.

The access modifier must be `public`.

```
public class HelloExam {
    public static void main(String args[]) {
        System.out.println("Hello exam");
    }
}
```

The nonaccess modifier must be `static`.

```
public static void main(String... args)
```

It is valid to define args as a variable argument

Comments in Java

Single Line Comment	//	Used to create a Single Line Comment	// System.out.println("Test");
Multi-Line Comment	/* … */	Used to comment a set of lines (block)	/* System.out.println("Line 1"); System.out.println("Line 2"); */

JavaDoc Comment	/** … */	Used to generate HTML based comments.	/**
			* @author JP <jp@jp.com>
			* @version 1.7
			* @since 2013-08-12
			*/
			public class TestDoc {
			}

Data Types

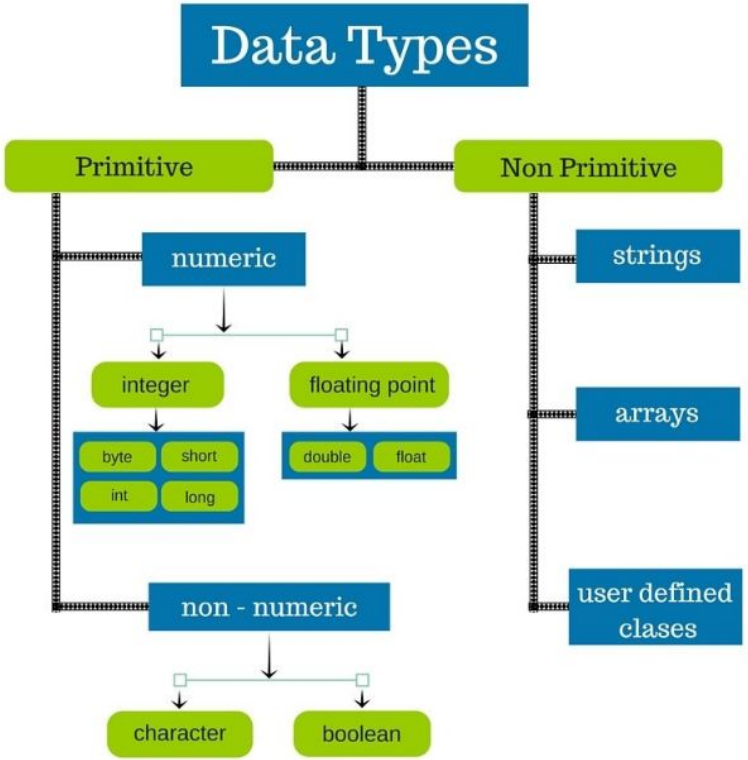
Java is a strongly typed language, which means every data or information has a type Known To Be Data Type that Data Type can not be changed once declared.

So every variable, literal or any other information has a type.

There are different Data Types to store various types of information.

These Data Type are broadly classified into:

- ✔ **Primitive data types:** These eight data types represent the building blocks for Java objects, because all Java objects are just a com-plex collection of these primitive data types.
- ✔ **Non primitive:** A reference type refers to an object (an instance of a class).



Examples of Data Types

Variable name

```
boolean result = false;
```

Type of variable

boolean literal

```
char c1 = 'D';
```

Use single quotes to assign a char, not double quotes

```
char c1 = 122;
```

Assign z to cl

```
char c3 = -122;
```

Fails to compile

```
byte num = 100;
short sum = 1240;
int total = 48764;
long population = 214748368;
```

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	[-2^{63}, $2^{63}-1$]	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

```
float average = 20.129F;
float orbit = 1765.65f;
double inclination = 120.1762;
```

Operators

Operator type	Operators	Purpose
Assignment	=, +=, -=, *=, /=	Assign value to a variable
Arithmetic	+, -, *, /, %, ++, --	Add, subtract, multiply, divide, and modulus primitives
Relational	<, <=, >, >=, ==, !=	Compare primitives
Logical	!, &&, ||	Apply NOT, AND, and OR logic to primitives

Let's have a look at some valid lines of code:

```
double myDouble2 = 10.2;

int a = 10;
int b = a;

float float1 = 10.2F;
float float2 = float1;
```

OK to assign variables of same type

OK to assign literal 10.2 to variable of type double

OK to assign literal 10 to variable of type int

OK to assign literal 10.2F to variable of type float

OK to assign variables of same type

Assignment Operators

Operator	Purpose	Example	Equivalent
+=	Addition	x += 2	x = x + 2
-=	Subtraction	x -= 2	x = x - 2
/=	Division	x /= 2	x = x / 2
*=	Multiplication	x *= 2	x = x * 2
%=	Modulus	x %= 2	x = x % 2

The simple assignment operator, =, is the most frequently used operator. It's used to initialize variables with values and to reassign new values to them

```
b += a;

a = b = 10;
b -= a;

a = b = 10;
b *= a;

a = b = 10;
b /= a;
```

Reassign a value of 10 to both variables a and b

OK; b is assigned a value of 20. b = 10 + 10.

OK; b is assigned a value of 10. b = 20 – 10.

b is assigned a value of 100. b = 10 * 10.

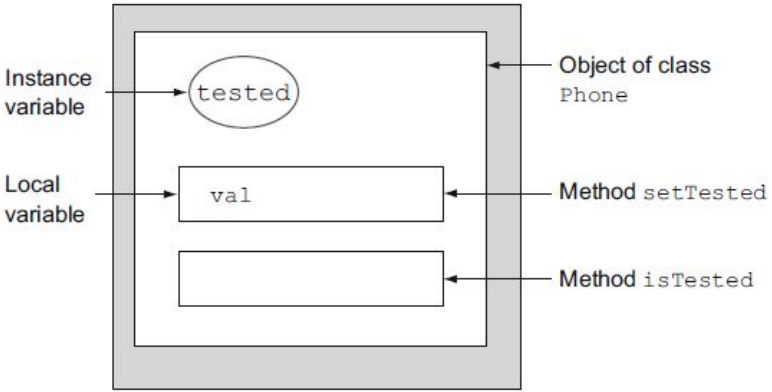
b is assigned a value of 1. b = 10 / 10.

Variables & Scope

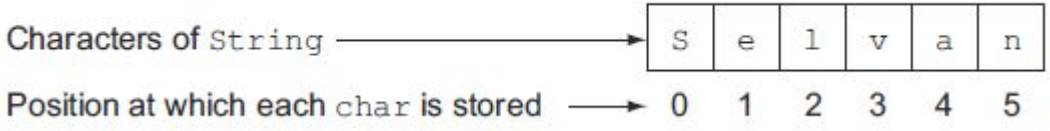
Instance Variables:

- *Instance* is another name for an object.
- Hence, an *instance variable* is available for the life of an object.
- Is declared within a class, outside of all methods.
- It's accessible to all the nonstatic methods defined in a class.

```
class Phone {
    private boolean tested;
    public void setTested(boolean val) {
        tested = val;
    }
    public boolean isTested() {
        return tested;
    }
}
```



Variables & Scope



Local Variables:

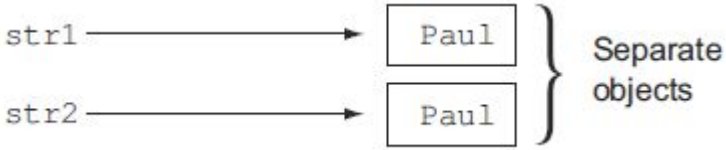
- ✔ The class String is defined in the Java API in the **java.lang** package.
- ✔ The String class represents character strings.

```
String str1 = new String("Paul");
String str2 = new String("Paul");
System.out.println(str1 == str2);
```

Create two String objects by using the operator new

When comparing the objects referred to by the variables str1 and str2, it prints false.

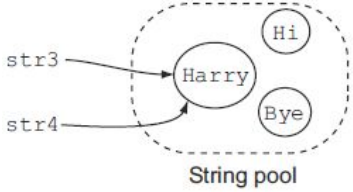
String objects created using the operator new always refer to separate objects, even if they store the same sequence of characters.




```
String str3 = "Harry";
String str4 = "Harry";
System.out.println(str3 == str4);
```

Create two String objects by using assignment operator =

Prints true because str3 and str4 refer to the same object



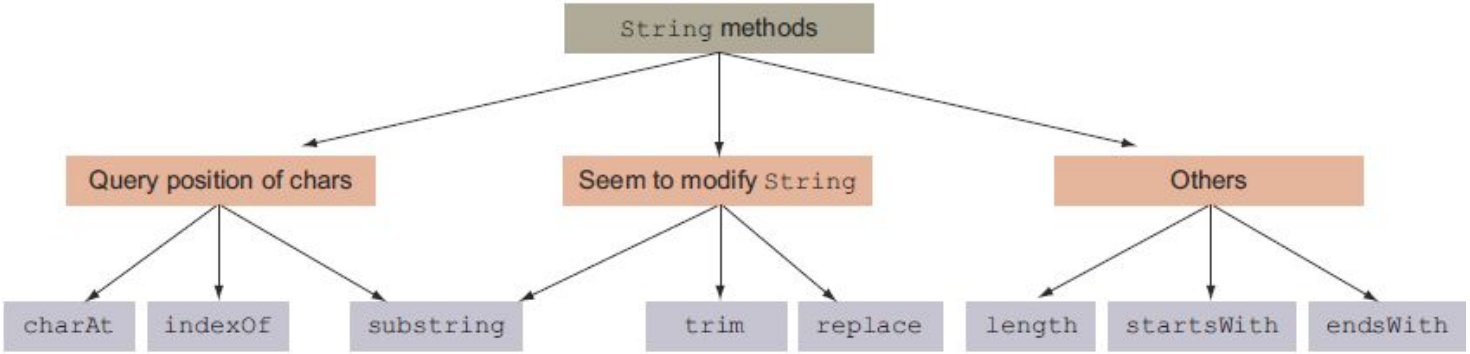
JAVA BASICS

Road Runners Training

CLASE III

Globant
we are ready

String class Methods



Quick Tip

Strings are **immutable**. Once initialized, a String value can't be modified. All the String methods that return a modified String value return a new String object with the modified value. The original String value always remains the same.

String class Methods

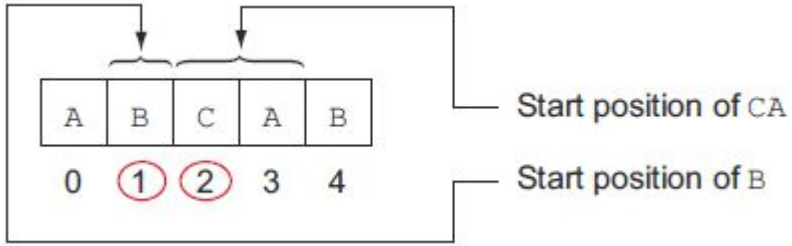
indexOf()

```
String letters = "ABCAB";
System.out.println(letters.indexOf('B'));
System.out.println(letters.indexOf("S"));
System.out.println(letters.indexOf("CA"))
```

Prints 1

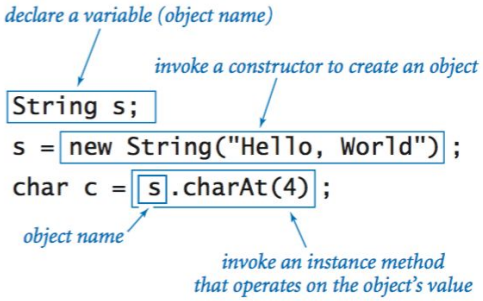
Prints –1

Prints 2




```
String letters = "ABCAB";
System.out.println(letters.indexOf('B', 2));
```

charAt()



startsWith() and endsWith()

```
String letters = "ABCAB";
System.out.println(letters.startsWith("AB"));
System.out.println(letters.startsWith("a"));
System.out.println(letters.startsWith("A", 3));
```

Prints true

Prints false

Prints true

```
System.out.println(letters.endsWith("CAB"));
System.out.println(letters.endsWith("B"));
System.out.println(letters.endsWith("b"));
```

Prints true

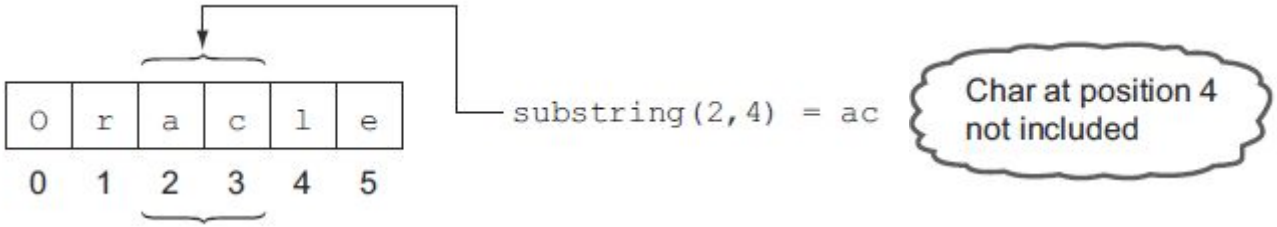
Prints true

Prints false

substring()

```
String exam = "Oracle";
String result = exam.substring(2, 4);
System.out.println(result);
```

Prints ac



length()

```
System.out.println("Shreya".length());
```

Other methods

contains(CharSequence text)
“Example”.contains(“amp”) >> true

equalsIgnoreCase(String text)
“Example”.equalsIgnoreCase(“example”) >> true

Returns a boolean true or false

length()
“Example”.length() >> 7

Returns an int

valueOf(...)
String.valueOf(12) >> “12”

Takes a value and returns it as a String

Constructors

Every class in Java has a constructor whether you code one or not. If you don't include any constructors in the class, Java will create a Default Constructor for you without any parameters.

```
public Rabbit() {}
```

The constructor of a class is called at the moment an object of that class is instantiated at compilation.

```
public class Rabbit {
  public static void main(String[] args) {
    Rabbit rabbit = new Rabbit();          // Calls default constructor
  }
}
```

Overloading constructors

You can have multiple constructors in the same class as long as they have different method signatures. The name is always the same since it has to be the same as the name of the class. This means constructors must have different parameters in order to be overloaded.

```
public class Hamster {
  private String color;
  private int weight;
  public Hamster(int weight) {                // first constructor
    this.weight = weight;
    color = "brown";
  }
  public Hamster(int weight, String color) {     // second constructor
   this.weight = weight;
   this.color = color;
  }
}
```

```
public Hamster(int weight) {
  this(weight, "brown");
}
```

Overloading constructors

```
public class Person {
    private String name;
    private int id;
    private int age;

    public Person() {
    }

    public Person(int i, String n) {
        id=i;
        name=n;
    }

    public Person(int i, String n, int a) {
        id=i;
        name=n;
        age=a;
    }

    public void display() {
        System.out.println(id+""+name+""+age);
    }

    public static void main(String args[]) {

        Person p1 = new Person(111,"Karan");
        Person p2 = new Person(222,"Aryan",25);
        p1.display();
        p2.display();

}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Setters and Getters

```
public class Person {

    private int id;
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

}
```

Getter: Its a method that, when called, returns the value of a variable.

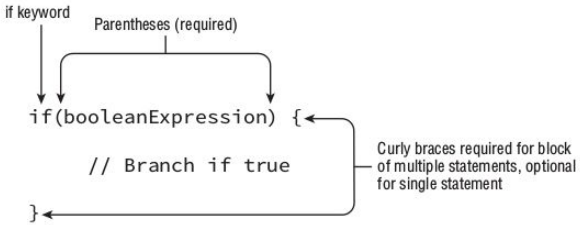
Setter: Its a method that, when called, sets the value of a variable.

The instance variables are usually set as private, while the setter and getter methods are set as **public** (which is part of the OO principle of Encapsulation), giving other classes the possibility of interacting with the given class, without exposing its methods and attributes publicly.

Control Flow Statements

Conditionals: If

The if-then statement, allows our application to execute a particular block of code if and only if a boolean expression evaluates to true at runtime.

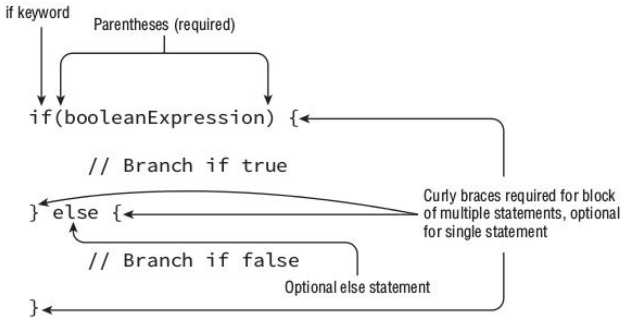



```
if(hourOfDay < 11) {
  System.out.println("Good Morning");
  morningGreetingCount++;
}
```

Control Flow Statements

Conditionals: If-Then-Else

Our code is branching between one of the two possible options, with the boolean evaluation happening only once.

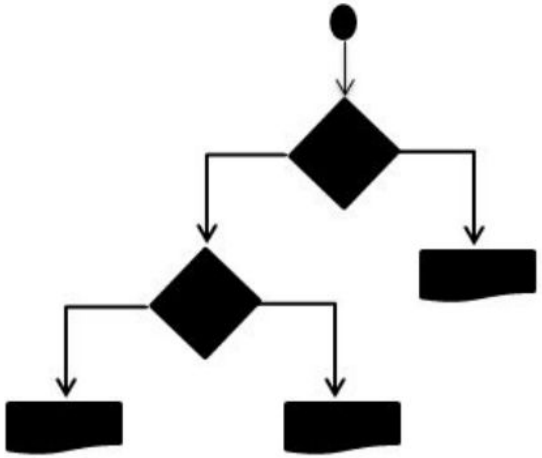



```
if(hourOfDay < 11) {
  System.out.println("Good Morning");
} else {
  System.out.println("Good Afternoon");
}
```

```
if(hourOfDay < 11) {
  System.out.println("Good Morning");
} else if(hourOfDay < 15) {
  System.out.println("Good Afternoon");
} else {
  System.out.println("Good Evening");
}
```

Control Flow Statements

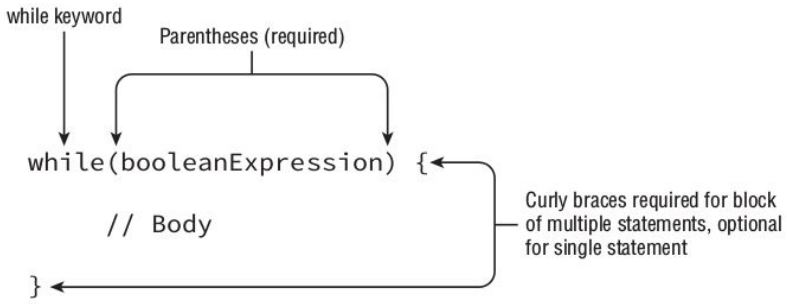
```
public class HelloWorld {

    public static void main(String[] args) {
        int number1 = 1;
        int number2 = 2;

        if (number1 == number2) {
            System.out.println("Should not enter here");
        }
        else if (number1 >= number2) {
            System.out.println("Should not enter here");
        } else {
            System.out.println("Should enter here");
        }
    }
}
```



Control Flow Statements

Loops (Iteration Statements): While

Executes a statement of code multiple times in succession.
For example, a statement that iterates over a list of unique names and outputs them.
The While loop has a termination condition, implemented as a boolean expression, that will continue as long as the expression evaluates to true.

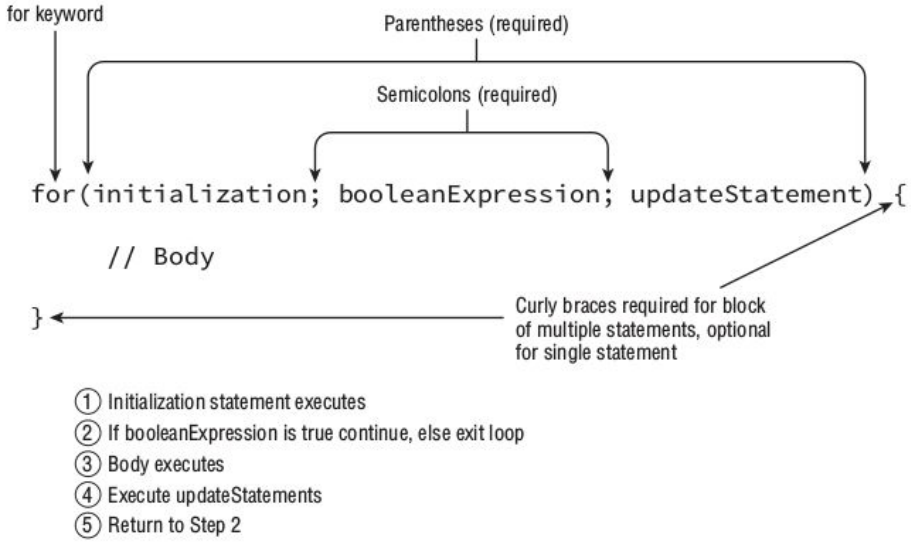



```
class WhileDemo
{
    public static void main(String arg[])
    {
        int x = 1;
        while( x <= 5 ) // LINE A
        {
            System.out.println("x = "+ x);
            x++;
        }

    }
}
```

Control Flow Statements

Loops (Iteration Statements): For

A basic for loop has the same conditional boolean expression and statement, or block of statements, as the other loops you have seen, as well as two new sections: an initialization block and an update statement.

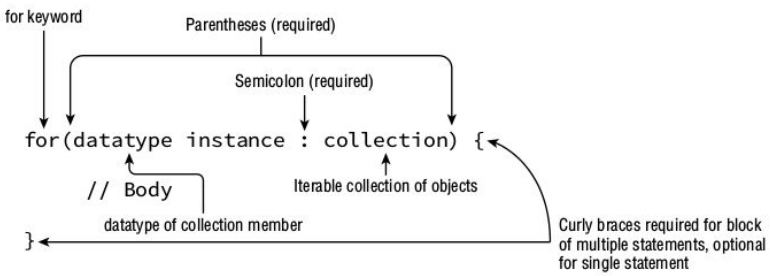



```java
class ForExample
{
    public static void main(String arg[])
    {
        for(int x = 0; x < 5; x++)
        {
            System.out.println("x = " + x);
        }
        System.out.println("After for loop");

    }
}
```

Control Flow Statements

Loops (Iteration Statements): For-Each

Specifically designed for iterating over arrays and Collection objects.




```
for(int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}
// becomes...
for(String s : array) {
   System.out.println(s);
}
```

Thank You