```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)
    {
        System.out.println(10+20);
        System.out.println("===>"+10+20);
    }
}
```

```
30
===>1020
```

After the string data, whenever you are writing + symbol, it will work as concatenation operation.

```java
package com.pack1;

public class ClassB
{
    public static void main(String[] args)
    {
        System.out.println(10+20);
        System.out.println("===>"+10+20);
        System.out.println("===>"+(10+20));
    }
}
```

```
30
===>1020
===>30
```

getClass()

It is going to provide fully qualified class name

```java
package com.pack1 ;

public class ClassA
{
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA();
        ClassA aobj2=new ClassA();

        System.out.println("getClass() : "+aobj1.getClass());
    }
}
/*
 getClass():   It is going to provide fully qualified Class Nar
 -----------
 */
```

```
getClass() : class com.pack1.ClassA
```

```java
package com.pack1 ;

public class ClassA
{
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA();
        ClassA aobj2=new ClassA();

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());
    }
}
/*
 getClass():   It is going to provide fully qualified Class Name
 -----------
 */
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA
```

\n is used to get a space line before in the console, this is a escape character. Our curser will be coming into the new line

toString()

It is going to convert our class object into a String format

The internal implementation of toString()

```
247      * value of:
248      * <blockquote>
249      * <pre>
250      * getClass().getName() + '@' + Integer.toHexString(hashCode())
251      * </pre></blockquote>
252      *
253      * @return  a string representation of the object.
254      */
255     public String toString() {
256         return getClass().getName() + "@" + Integer.toHexString(hashCode());
257     }
258
259     /**
260      * Wakes up a single thread that is waiting on this object's
261      * monitor. If any threads are waiting on this object, one of them
262      * is chosen to be awakened. The choice is arbitrary and occurs at
263      * the discretion of the implementation. A thread waits on an object's
264      * monitor by calling one of the {@code wait} methods.
265      * <p>
266      * The awakened thread will not be able to proceed until the current
267      * thread relinquishes the lock on this object. The awakened thread will
```

```
21
22  toString(): It is going to convert Our Class Object into a String format
23  ----------
24
25  public String toString()
26  {
27          return getClass().getName() + "@" + Integer.toHexString(hashCode());
28  }
29
30  //     com.pack1.ClassA@fdyd7g
31  */
```

The number present at right last called as Hexa integer value of our class object

```java
        ClassA aobj1=new ClassA();
        ClassA aobj2=new ClassA();

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

    }
}
/*
 getClass():   It is going to provide fully qualified Class Name
 -----------

 toString(): It is going to convert Our Class Object into a String format
 ----------

 public String toString()
 {
        return getClass().getName() + "@" + Integer.toHexString(hashCode(
 }

 //  aobj1 ===>  com.pack1.ClassA@fdyd7gk
 //  aobj2 ===>  com.pack.ClassA@Ausfu567
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a
```

Because of .getName() we are not getting class

finalize()

finalize() it will be called internally by the Garbage collector if there is an object eligible for destruction.

garbage collection

It is a process of destroying all unused or unreferenced objects from the heap area.

The garbage collector will be working in the background of every java program. You don't need to call it manually.

Whenever we are calling a garbage collector if there is an object eligible for destruction then only the garbage collector act on your program. Otherwise, it will just monitor the situation.

```java
public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA();
        ClassA aobj2=new ClassA();

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();



    }
}
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a

meth1() called
```

```java
public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA();
        ClassA aobj2=new ClassA();

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();

        System.gc(); // It is a method to call Garbage Collector manually

    }
}
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a

meth1() called
```

You cannot perform any operations on null.

true, false and null are not key words, those are values.

```java
public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();

        aobj1=null;

        System.gc(); // It is a method to call Garbage Collector manually

    }
}
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a

meth1() called
```

```java
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    protected void finalize()
    {
        System.out.println("Garbage has been collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();

        //aobj1=null;

        System.gc(); // It is a method to call Garbage Collector manually
    }
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a

meth1() called
```

```java
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    protected void finalize()
    {
        System.out.println("Garbage has been collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();

        aobj1=null;

        System.gc(); // It is a method to call Garbage Collector manually
    }
```

```
aobj1 getClass() : class com.pack1.ClassA
aobj2 getClass() : class com.pack1.ClassA

aobj1 toString()) : com.pack1.ClassA@10f87f48
aobj2 toString()) : com.pack1.ClassA@b4c966a

meth1() called
Garbage has been collected
```

```java
public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
    }
    protected void finalize()
    {
        System.out.println("Garbage has been collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 getClass() : "+aobj1.getClass());
        System.out.println("aobj2 getClass() : "+aobj2.getClass());

        System.out.println("\naobj1 toString()) : "+aobj1.toString());
        System.out.println("aobj2 toString()) : "+aobj2.toString());

        aobj1.meth1();

        aobj1=null;
```

```java
        aobj1=null;

        System.gc(); // It is a method to call Garbage Collector manually
    }
}
/*
 getClass():   It is going to provide fully qualified Class Name
 -----------

 toString(): It is going to convert Our Class Object into a String format
 ----------

 public String toString()
 {
        return getClass().getName() + "@" + Integer.toHexString(hashCode());
 }

 finalize():
 ------------
 1) finalize() will be called internally by the Garbage Collector
    if there is an Object eligible for destruction
 */
```

Object:

- Objects are created from a class
- To create an object of a Class, specify the class name, followed by the object/reference name, and use the keyword "**new**" (We can initialize that object by using constructors).
- We can create multiple objects of one class.

Note: **reference** is the address of the memory location where the object is stored

Variable:

- A variable provides identity to memory location
- Using variables we can process the information easily
- Variables can also be called as References & Identifiers

OBJECT

- An object is created from a class.
- To create an object, first we need to write a new key word and initialize that object with the help of the constructor and constructor name should be same as class name.
- We can create multiple objects of a class.

Variables

- ❖ A variable provides identity to a memory location.
- ❖ Using variables, we can process information easily.

Naming Conventions of a Java Identifier:

1) A java identifier can start with
   - A to Z
   - a to z
   - $ (or) _
2) A Java identifier can **never** starts with a number, but we can use any number **combinations** between 0 to 9 in the identifiers.
3) Only '2' special characters are allowed in the identifiers **$** and **_**
4) Spaces are NOT allowed in the identifiers.
5) We can take any length as identifier length.
6) As java is case-senstive int a=10; and int A=20; both are NOT SAME
7) All the Java language keywords we cant use them as Identifiers
8) ***We can use Class names as identifiers, but it is highly not recommended***

```
public class ClassA
{
    void meth1()
    {
        int x=10;

        System.out.println(x);
    }

    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();

        aobj.meth1();
    }
}
```

Identifier

Understanding Identifier :

- **A name in JAVA program is called identifier.**
- It may be class name, method name, variable name.

Rules [8]:

✓ The only allowed characters in java identifiers are:

1) a to z
2) A to Z
3) 0 to 9
4) _(underscore) and $

✓ If we are using any other symbols we will get compile time error.

✓ We can't start a JAVA Identifier with number.

✓ Java Identifiers are case sensitive (Actually JAVA itself is a case sensitive)

✓ We can take our own length for the name of an JAVA identifier.

✓ We can't use JAVA language keywords (50) as identifiers.

✓No space is allowed between the characters of an identifier

✓ All predefined JAVA class names and interface names we can use as identifiers **(X)**

```
int String =10;
String s="Java";
```

This an example for point 8

It is good programming practice that your identifiers' name should reflect their purpose.

All the 8 primitive data types are key words not valid for identifier name