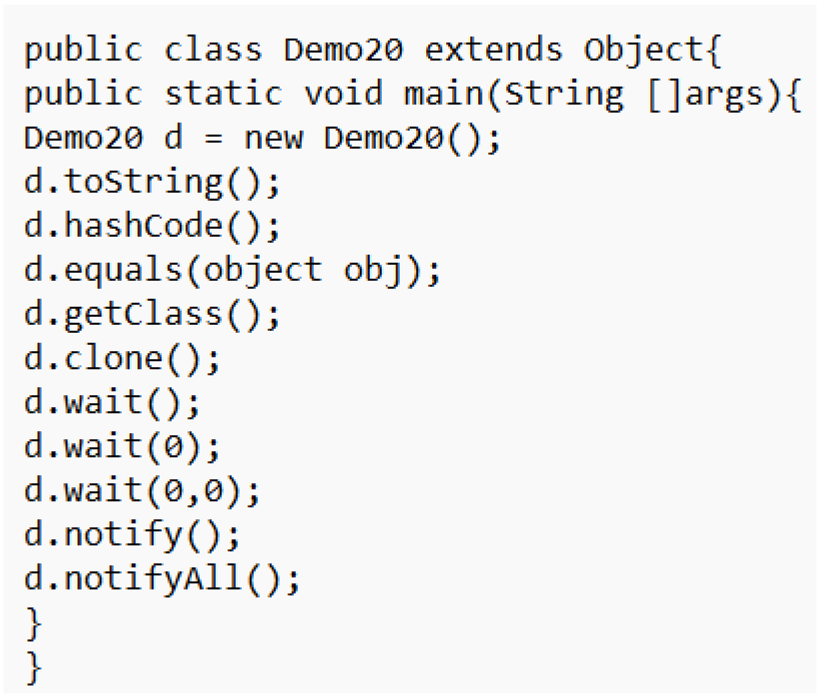
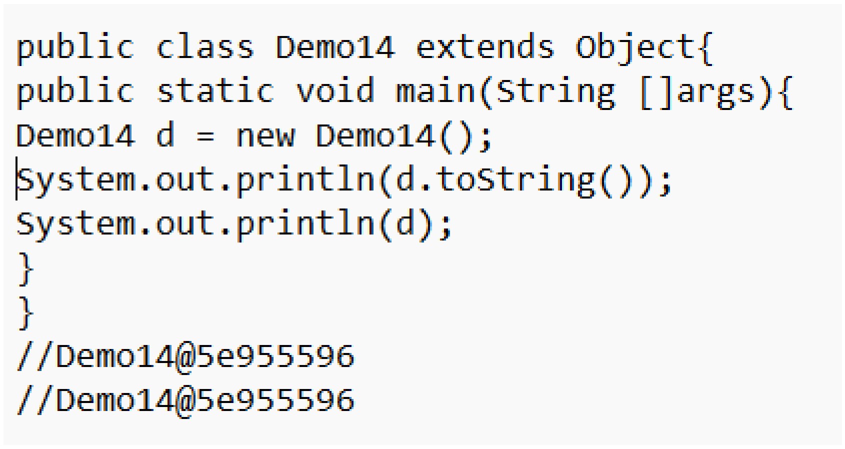
JAVA LIBRARY

- It is collection of pre-defined JAR (java archive files)
- JAR files are collection of packages. Packages are collection of classes and Class is a collection of pre-defined variables, methods, and constructors.
- OBJECT CLASS:
- It is the super most or parent most or route class for all the classes in java.
- It is a pre-defined class which is defined in java.lang package.
- Object class contains only one constructor i.e. no argument constructor.
- It contains 11 non-static reusable methods. They are



```
public class Demo20 extends Object{
public static void main(String []args){
Demo20 d = new Demo20();
d.toString();
d.hashCode();
d.equals(object obj);
d.getClass();
d.clone();
d.wait();
d.wait(0);
d.wait(0,0);
d.notify();
d.notifyAll();
}
}
```

- 1. toString ()
  - It is a pre-defined method in Object class
  - It represents Fully Qualified Class Name: Packagename.className@address of object.



```
public class Demo14 extends Object{
public static void main(String []args){
Demo14 d = new Demo14();
System.out.println(d.toString());
System.out.println(d);
}
}
//Demo14@5e955596
//Demo14@5e955596
```

  -
  - Here we didn't use package so the output doesn't contain it.
  - By default, it will invoke toString () method.
  - It returns String as an output.

- We can override this method like

```
public class Demo14 extends Object{
public String toString(){
return "Overridden in Demo14 class";
}
public static void main(String []args){
Demo14 d = new Demo14();
System.out.println(d.toString());
System.out.println(d);
}
}
//Overridden in Demo14 class
//Overridden in Demo14 class
```

- Here extends Object class is just to know, but it is default.

■ 2. hashCode ()

- It is a pre-defined method which is defined in Object class.
- It is used to auto generate Random integer value.

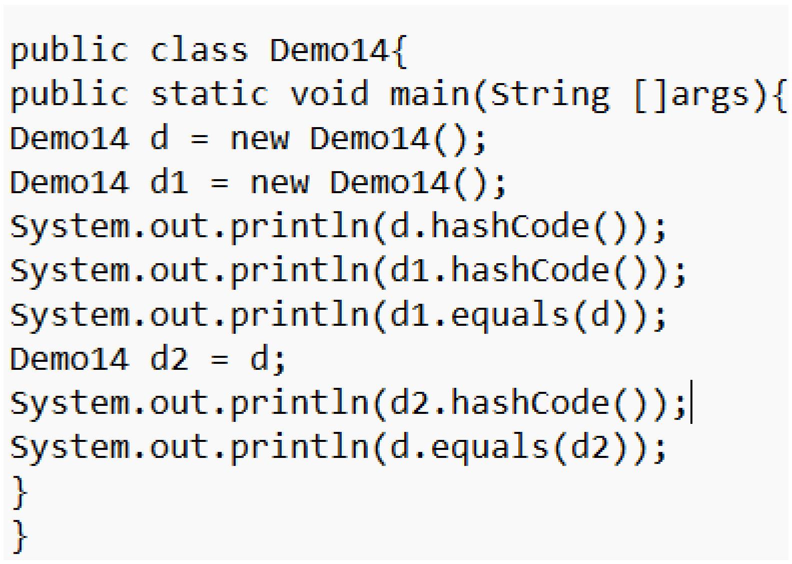
```
public class Demo14{
public static void main(String []args){
Demo14 d = new Demo14();
Demo14 d1 = new Demo14();
System.out.println(d.hashCode());
System.out.println(d1.hashCode());
}
}
//1586845078
//1356728614
```

- It returns int value.

```
public class Demo14{
public static void main(String []args){
Demo14 d = new Demo14();
Demo14 d1 = d;
System.out.println(d.hashCode());
System.out.println(d1.hashCode());
}
}
//480971771
//480971771
```

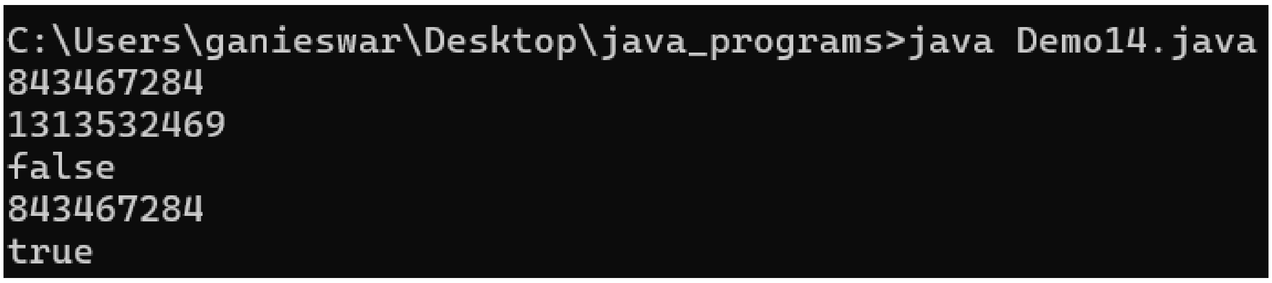
-
- It gives same integer value for same object reference.
- We can also override hashCode () method by
  return Random().nextInt();

- 3.equals ()
  - It is a pre-defined method which is defined in Object class.
  - It is used to compare hashCode (Address) of the Objects.
  - It returns Boolean type.



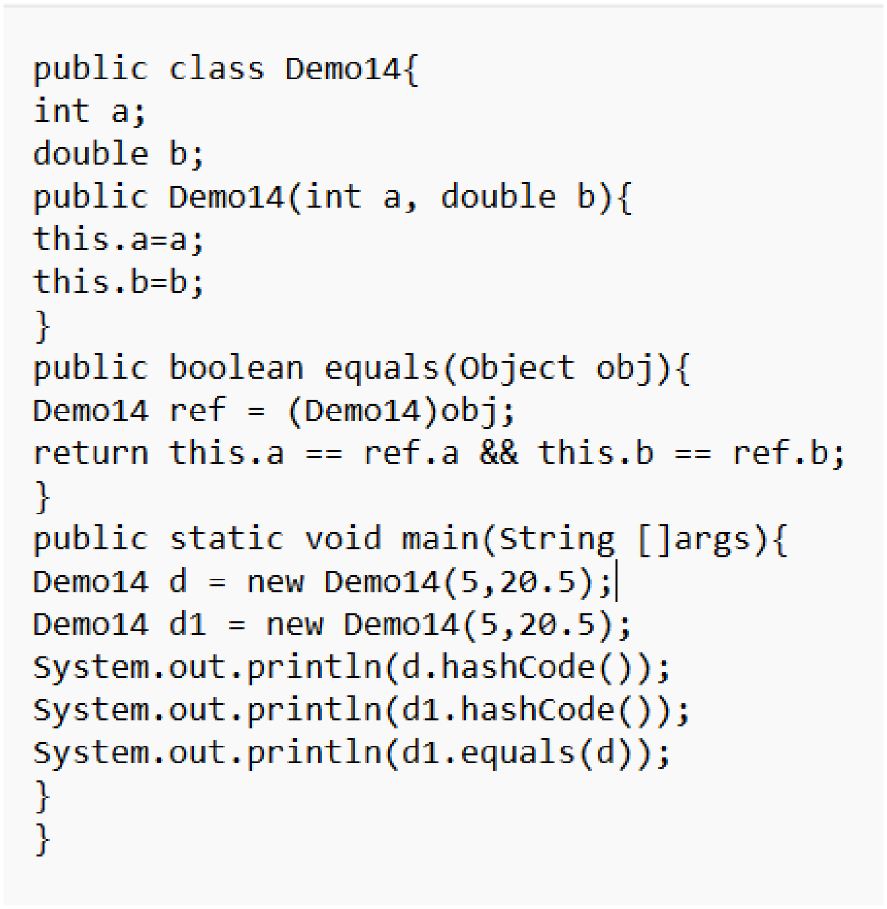
```java
public class Demo14{
public static void main(String []args){
Demo14 d = new Demo14();
Demo14 d1 = new Demo14();
System.out.println(d.hashCode());
System.out.println(d1.hashCode());
System.out.println(d1.equals(d));
Demo14 d2 = d;
System.out.println(d2.hashCode());
System.out.println(d.equals(d2));
}
}
```

  -



```
C:\Users\ganieswar\Desktop\java_programs>java Demo14.java
843467284
1313532469
false
843467284
true
```

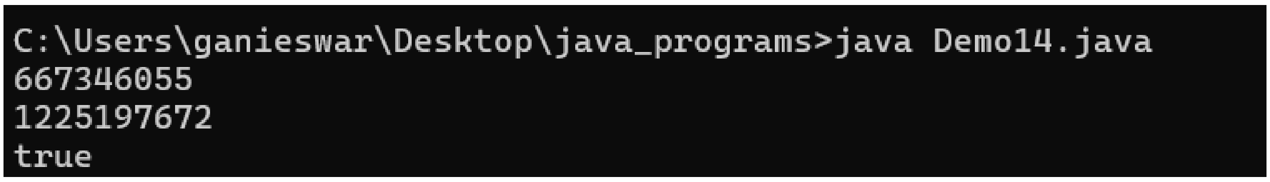
  -
  - We can override equals methods and improves it like not comparing the hashCode but the values.



```java
public class Demo14{
int a;
double b;
public Demo14(int a, double b){
this.a=a;
this.b=b;
}
public boolean equals(Object obj){
Demo14 ref = (Demo14)obj;
return this.a == ref.a && this.b == ref.b;
}
public static void main(String []args){
Demo14 d = new Demo14(5,20.5);
Demo14 d1 = new Demo14(5,20.5);
System.out.println(d.hashCode());
System.out.println(d1.hashCode());
System.out.println(d1.equals(d));
}
}
```

  -

-



```
C:\Users\ganieswar\Desktop\java_programs>java Demo14.java
667346055
1225197672
true
```

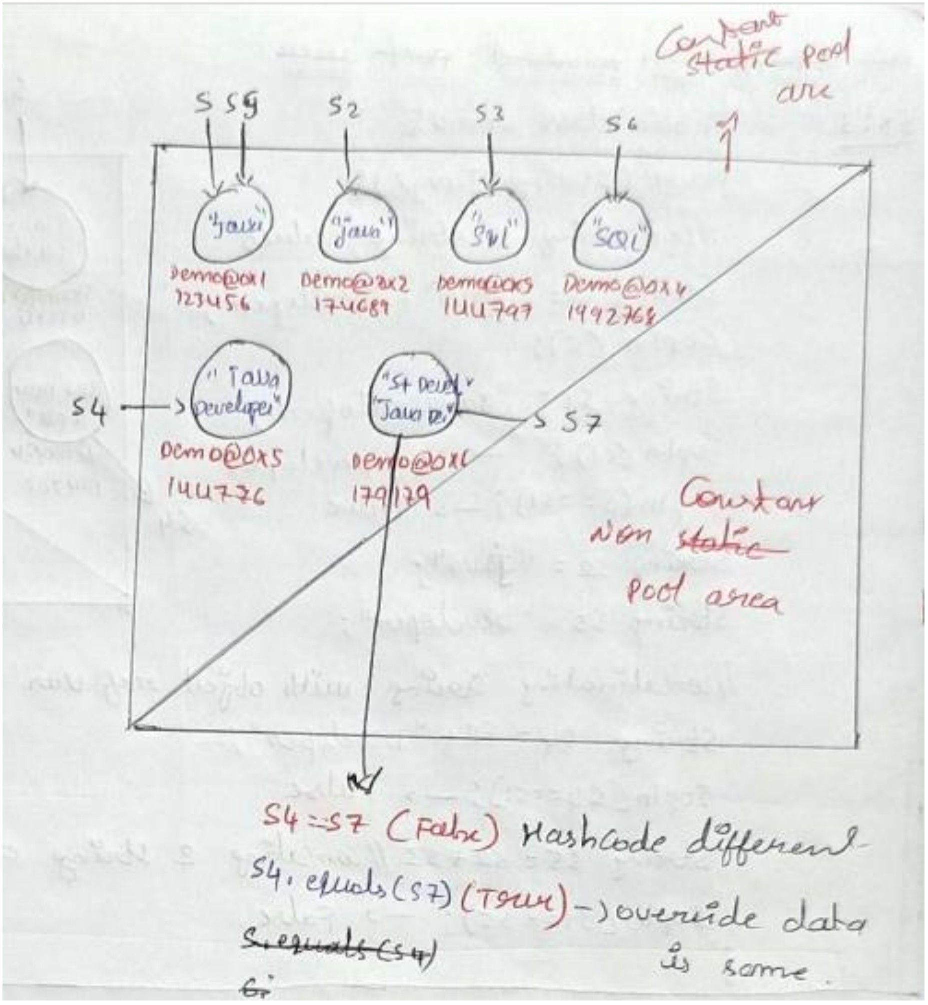
- Here the hashCode are different but the values in it are equals so it returns true.
- Even though hashCode is different equals () return true because it is overridden and comparing the data instead of hashCode.

■ STRING CLASS

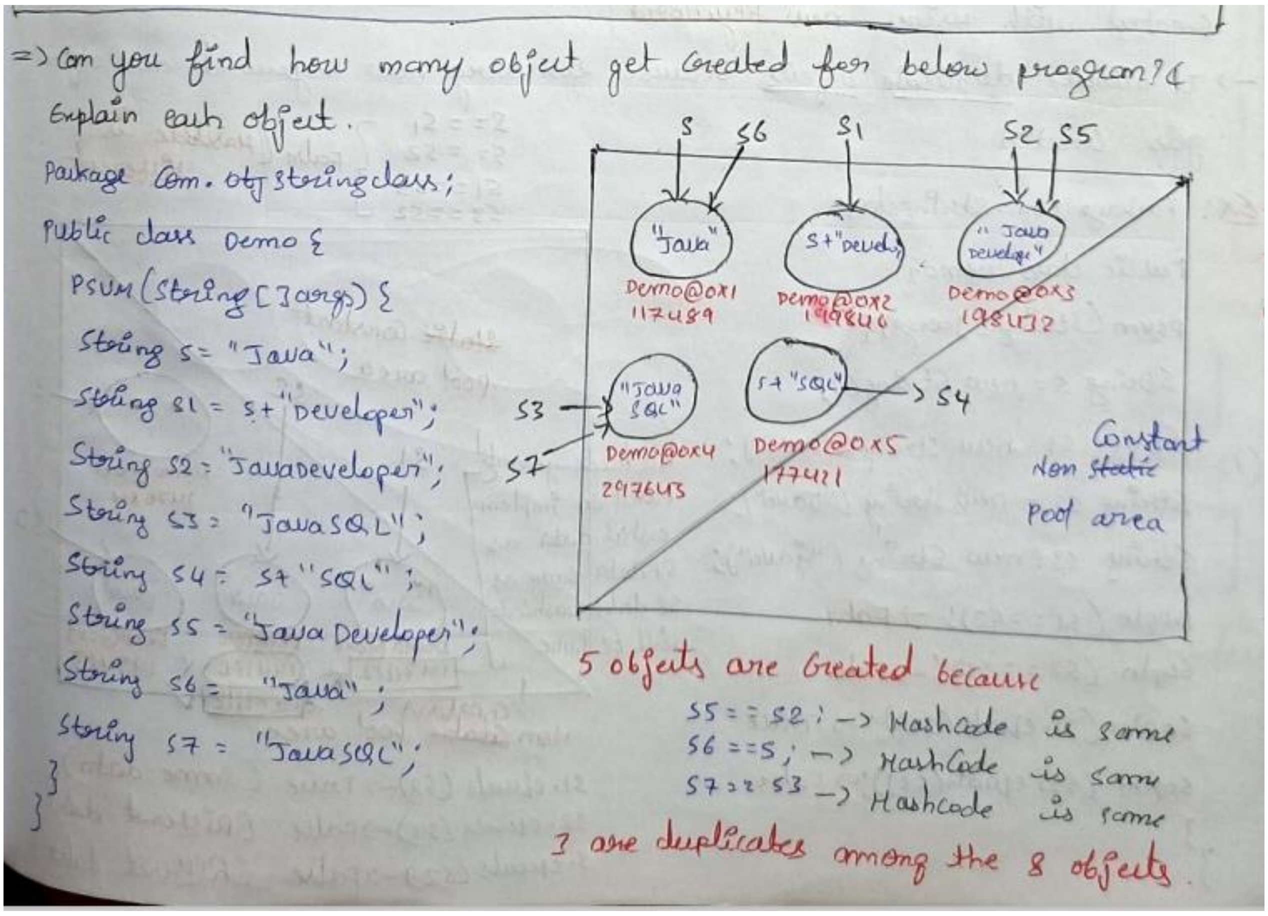
- String class is a pre-defined class which is defined in java.lang package.
- String class is a collection of characters which is enclosed with " ".
- Default value is Null, it is immutable class (unmodified) and synchronized class.
- We can create objects for string class in two ways:
- 1. Without using new keyword or String literal: String s = "ganesh";
- 2. With using new keyword: String s = new String (ganesh);
- String class objects will be stored in String pool area.
- String pool area is divided into two types: String Constant pool area and String Non-Constant pool area.
- **String Constant pool area** is used to store the object which is created without using new keyword.
- String Constant pool area won't allow duplicate objects.
- When object creation is declared, internally equal () method called and it compares an object's string literal, if equals () method returns true it means the object is already created in String Constant pool area. So, it won't allow to create a duplicate object instead of that it will refer the object which is already created with the same implemented data.
- For Example: String s1 = "java";
  String s2=" ";
  String s3= "java";
  s1== s2 (False)   s2==s3 (False)   s1==s3(True)
  s1.equals (s2) is False
  s1.equals (s3) is true
- Here both the s1 and s3 are pointing same string even though different variables.
- **In String class toString (), hashCode () and equals () are already overridden.**

```
public class Demo15{
public static void main(String []args){
String s1="Java";
String s2="java";
String s3="sql";
String s4="JavaDeveloper";
String s5="Java";
String s6="SQL";
System.out.println(s1==s2);//false
System.out.println(s3==s6);//false
System.out.println(s1==s5);//true
String s7 = s1+"Developer";
System.out.println(s7);//JavaDeveloper
System.out.println(s1==s2);//false
System.out.println(s4==s7);//false
System.out.println(s1.equals(s5));//true
System.out.println(s4.equals(s7));//true
}
}
```

-
- Here s7 and s4 are values equals but hashCode will different, to understand more about it:



S4==S7 (False) Hashcode different
S4.equals(S7) (True) -> override data is same
~~S.equals(S4)~~
6.



=) Can you find how many object get Created for below program?(
Explain each object.

```
Package Com.otj stringclass;
Public class Demo {
  PSVM(String[]args) {
   String s= "Java";
   String s1 = s+"Developer";
   String s2= "JavaDeveloper";
   String s3= "JavaSQL";
   String s4= s+"SQL";
   String s5 = "Java Developer";
   String s6= "Java";
   String s7 = "JavaSQL";
  }
}
```

s
s6
s1
s2
s5
"Java"
Demo@0x1 117489
s+"Devel"
Demo@0x2 199846
"Java Develope"
Demo@0x3 198432
s3
s7
"Java SQL"
Demo@0x4 297643
s+"SQL"
s4
Demo@0x5 177421
Constant
Non ~~Static~~
Pool area

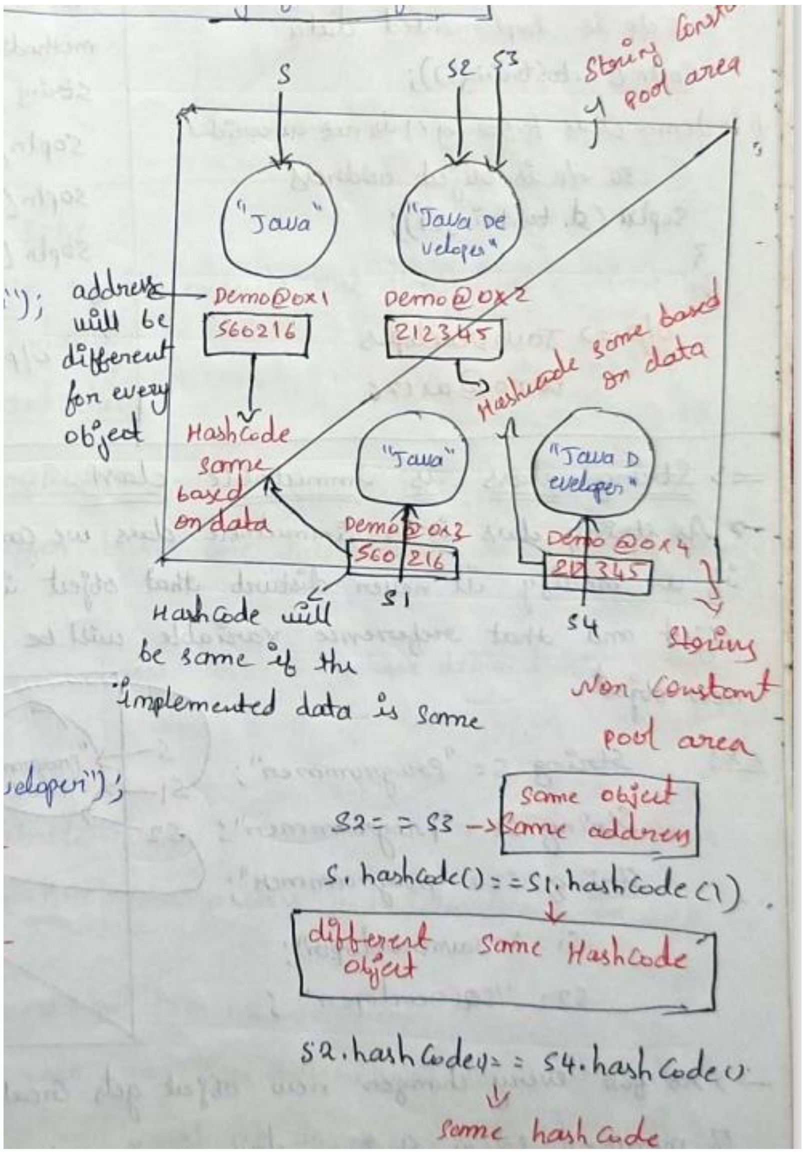
5 objects are Created because
s5 == s2 ; -> Hashcode is same
s6 == s ; -> HashCode is same
s7 == s3 -> Hashcode is same
3 are duplicates among the 8 objects.

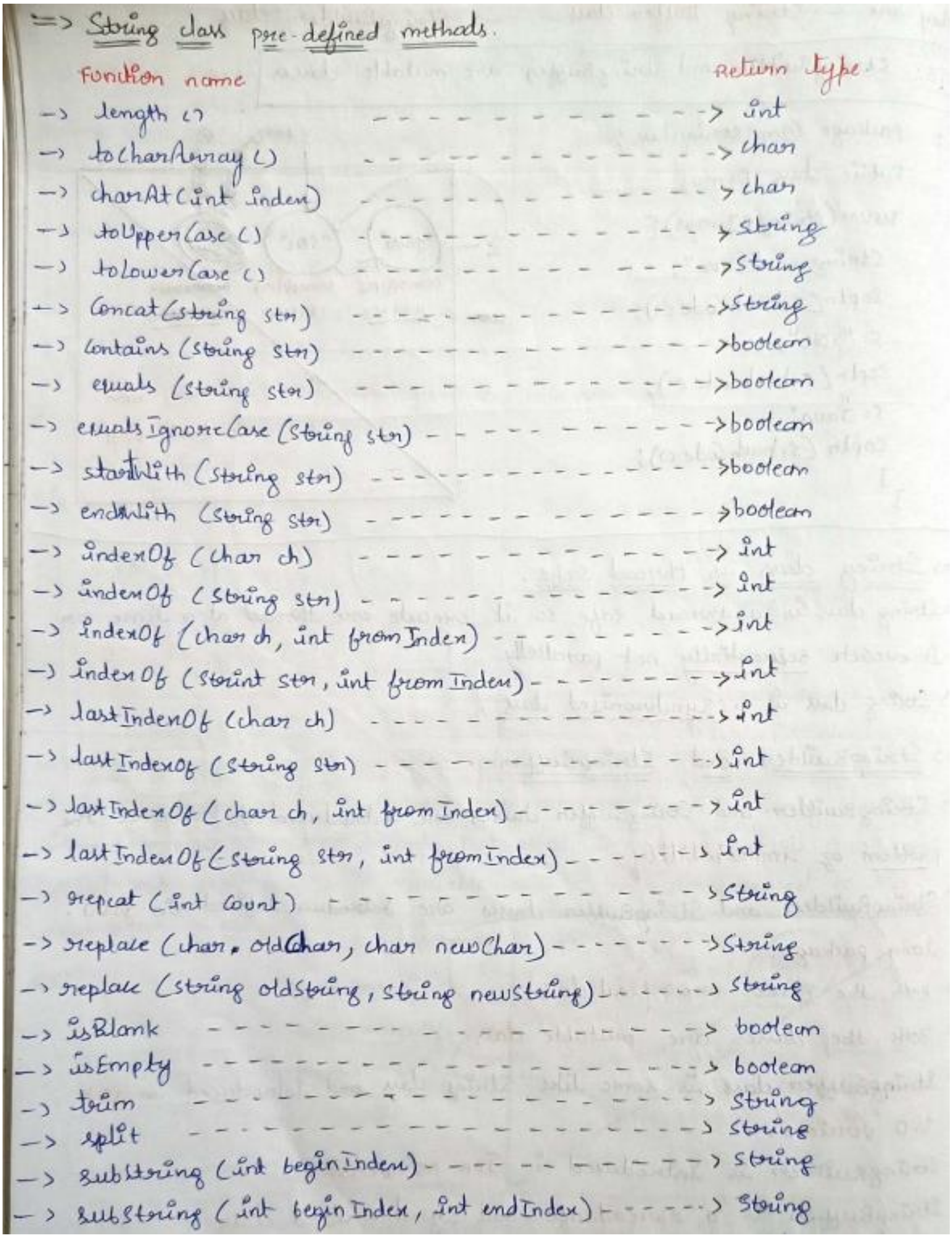
-
- **String non constant pool area** is used to store an object which is created with using **new keyword.**
- It allows duplicates objects because for every new keyword new object gets created.

```
public class Demo15{
public static void main(String []args){
String s1=new String("Java");
String s2=new String("Java");
String s3=new String("java");
System.out.println(s1==s2);//false
System.out.println(s1==s3);//false
System.out.println(s1.equals(s2));//true
System.out.println(s2.equals(s3));//false
}
}
```

- Here s1 and s2 are different objects so gave false as a result, but they have same values so equals gives true as output.
- == operator checks the address of the object.
- Address will be different for every object but hashCode is same for those data is equal. (exception is Strings using + concatenation)

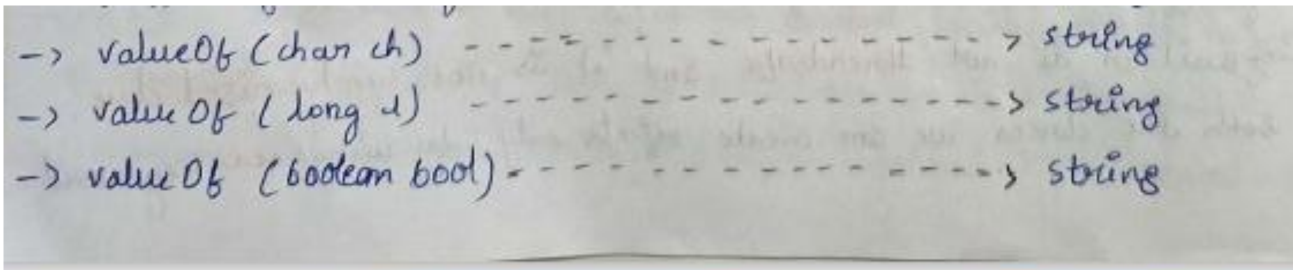
```
public class Demo15{
public static void main(String []args){
String s = "Java";
String s1=new String("Java");
System.out.println(s.toString());//Java
System.out.println(s.hashCode());//2301506
System.out.println(s1.hashCode());//2301506
System.out.println(s==s1);//false
System.out.println(s.equals(s1));//true
String s2 = "JavaDeveloper";
String s3 = "JavaDeveloper";
String s4 =new String("JavaDeveloper");
System.out.println(s2.hashCode());//-2146380888
System.out.println(s3.hashCode());//-2146380888
System.out.println(s4.hashCode());//-2146380888
System.out.println(s2==s3);//true
System.out.println(s2==s4);//false
System.out.println(s2.equals(s3));//true
System.out.println(s3.equals(s4));//true
}
}
```


String Constant pool area
s
s2
s3
"Java"
"Java Developer"
address will be different for every object
Demo@0x1
560216
Demo@0x2
212345
HashCode same based on data
Hashcode same based on data
"Java"
"Java Developer"
Demo@0x3
560216
Demo@0x4
212345
s1
s4
HashCode will be same if the implemented data is same
String Non Constant pool area
s2==s3 -> Same object Same address
s.hashcode()==s1.hashcode()
different object Same HashCode
s2.hashCode()==s4.hashCode()
same hash code

- As string class is immutable class, we cannot modify String class and we need to create new objects so it consumes a lot of memory, To overcome this problem there are two updates classes; they are: String Buffer class and String Builder class.
- These classes are defined in lang package.
- String Buffer is a threadsafe and synchronized class.
- String Builder is not threadsafe and not synchronized class.
- 

==> String class pre-defined methods.

Function name	Return type
--> length ()	int
--> toCharArray ()	char
--> charAt(int index)	char
--> toUpperCase ()	String
--> toLowerCase ()	String
--> Concat(String str)	String
--> Contains (String str)	boolean
--> equals (String str)	boolean
--> equalsIgnoreCase (String str)	boolean
--> startWith (String str)	boolean
--> endWith (String str)	boolean
--> indexOf (char ch)	int
--> indexOf (String str)	int
--> indexOf (char ch, int fromIndex)	int
--> indexOf (Strint str, int fromIndex)	int
--> lastIndexOf (char ch)	int
--> lastIndexOf (String str)	int
--> lastIndexOf (char ch, int fromIndex)	int
--> lastIndexOf (String str, int fromIndex)	int
--> repeat (int count)	String
--> replace (char oldChar, char newChar)	String
--> replace (String oldString, String newString)	String
--> isBlank	boolean
--> isEmpty	boolean
--> trim	String
--> split	String
--> subString (int beginIndex)	String
--> subString (int beginIndex, int endIndex)	String



-> valueOf (char ch) - - - - - - - - - - - - - - - - > string
-> value Of (long l) - - - - - - - - - - - - - - - - - - > string
-> value Of (boolean bool) - - - - - - - - - - - - - - - > string

-

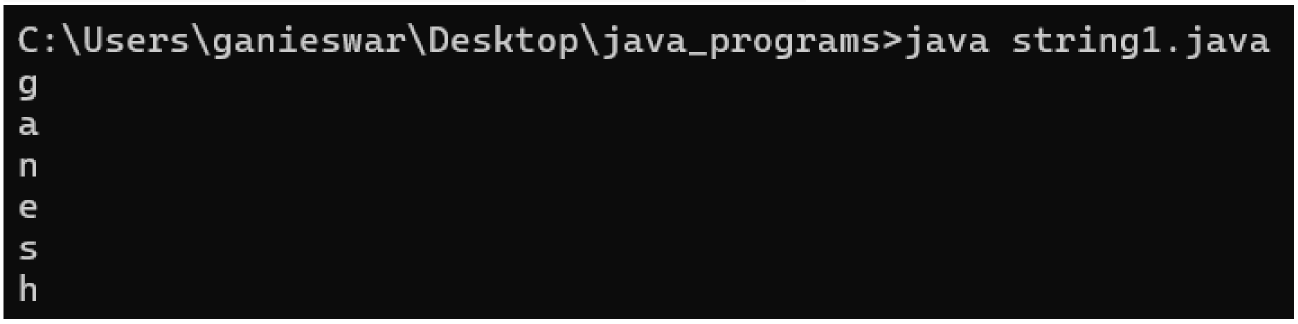
-> value Of (int i) - - - - - - - - - - - - - - - - - - > String
-> value Of (float f) - - - - - - - - - - - - - - - - - > String
-> Value Of (double d) - - - - - - - - - - - - - - - - - > string
-> compareTo (string) - - - - - - - - - - - - - - - - - > int

- Finding length of a string

```
public class string1{
public static void main(String []args){
String s ="ganesh";
System.out.println(s.length());
}
}
```
//6

- Converting String into character array

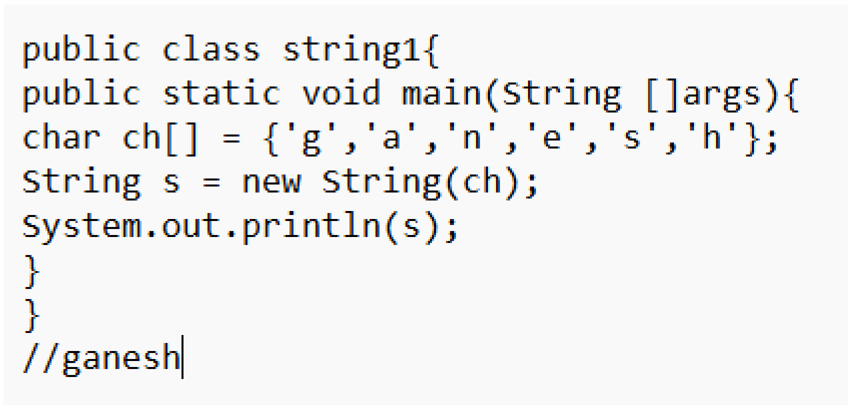
```
public class string1{
public static void main(String []args){
String s ="ganesh";
char ch[] = s.toCharArray();
for(int i=0;i<ch.length;i++){
System.out.println(ch[i]);
}
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java string1.java
g
a
n
e
s
h
```

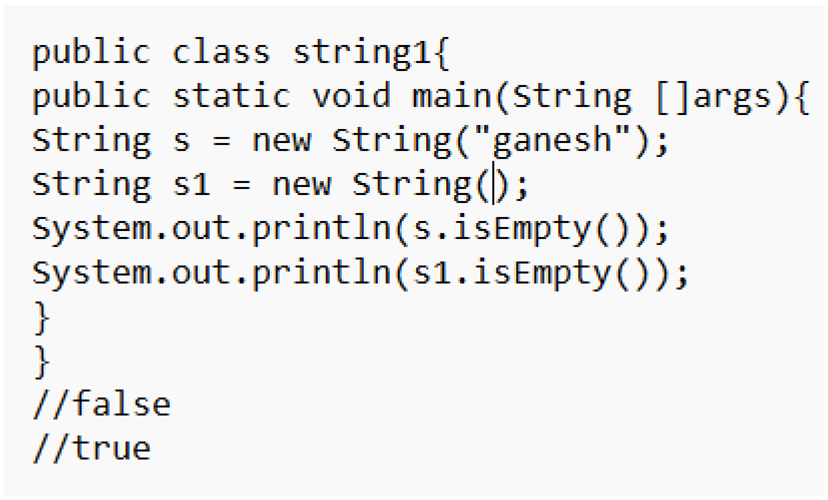
Here ch.length as the condition for termination.

- Converting character array to string



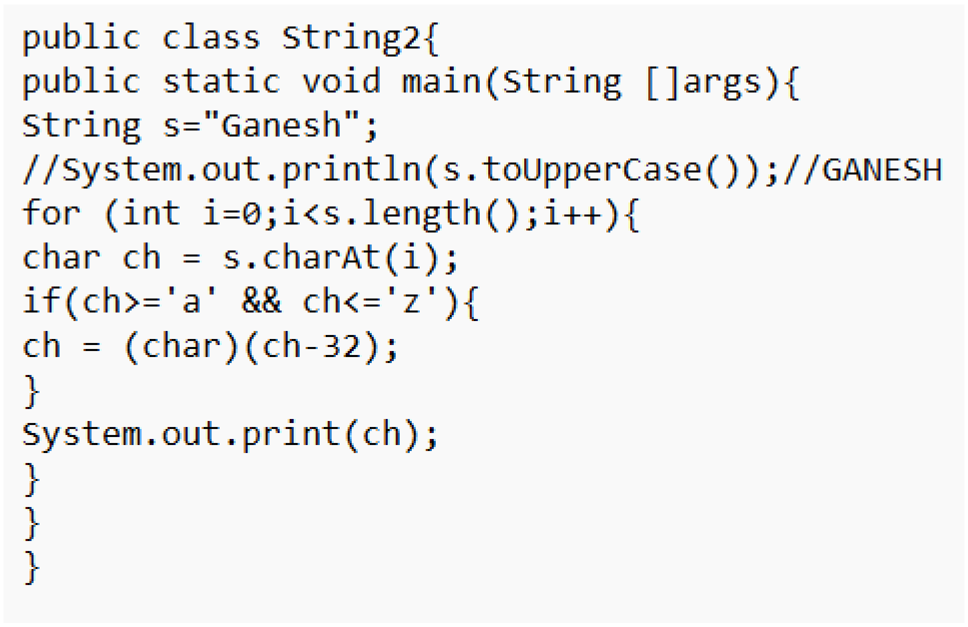
```
public class string1{
public static void main(String []args){
char ch[] = {'g','a','n','e','s','h'};
String s = new String(ch);
System.out.println(s);
}
}
//ganesh
```

- Checking whether the String is empty or not?



```
public class string1{
public static void main(String []args){
String s = new String("ganesh");
String s1 = new String();
System.out.println(s.isEmpty());
System.out.println(s1.isEmpty());
}
}
//false
//true
```

- Converting LowerCase to UpperCase:

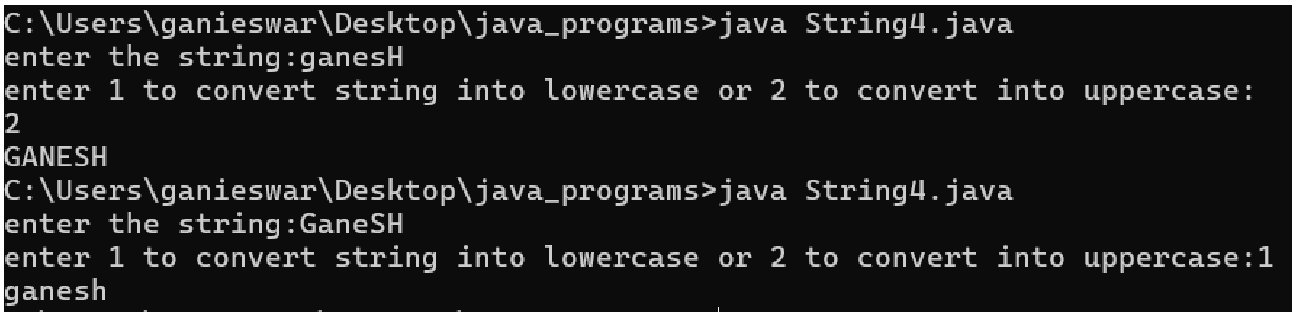


```
public class String2{
public static void main(String []args){
String s="Ganesh";
//System.out.println(s.toUpperCase());//GANESH
for (int i=0;i<s.length();i++){
char ch = s.charAt(i);
if(ch>='a' && ch<='z'){
ch = (char)(ch-32);
}
System.out.print(ch);
}
}
}
```

-
- Here we can see by using toUpperCase method and by building the logic.
- //65 to 90 (A); 97 to 122 (a) So, to convert into UpperCase we reduce its value by 32.
- Type casting is done here (char) (ch-32), () are important for both.
- We only use print instead of println to have the output as one line.
- And condition in the for loop we use s.length () method mean while we use ch.length if it is a character array.

```
import java.util.Scanner;
public class String4{
public static void main(String []args){
Scanner sc= new Scanner(System.in);
System.out.print("enter the string:");
String s = sc.nextLine();
System.out.print("enter 1 to convert string into lowercase
or 2 to convert into uppercase:");
int n=sc.nextInt();
switch(n){
case 1 : lowercase(s);
break;
case 2 : uppercase(s);
break;
}

}
public static void lowercase(String s){
for(int i=0;i<s.length();i++){
char ch = s.charAt(i);
if(ch>='A' && ch<='Z'){
ch = (char)(ch + 32);
}
System.out.print(ch);
}
}
public static void uppercase(String s){
for(int i=0;i<s.length();i++){
char ch = s.charAt(i);
if(ch>='a' && ch<='z'){
ch = (char)(ch - 32);
}
System.out.print(ch);
}
}

}
```

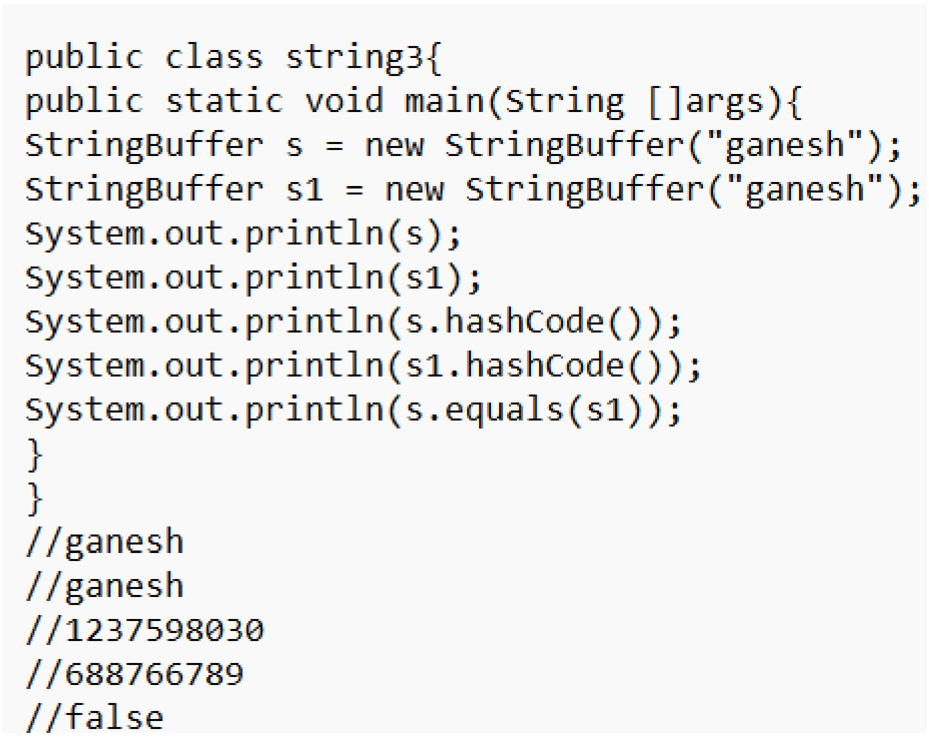
-



```
C:\Users\ganieswar\Desktop\java_programs>java String4.java
enter the string:ganesH
enter 1 to convert string into lowercase or 2 to convert into uppercase:
2
GANESH
C:\Users\ganieswar\Desktop\java_programs>java String4.java
enter the string:GaneSH
enter 1 to convert string into lowercase or 2 to convert into uppercase:1
ganesh
```

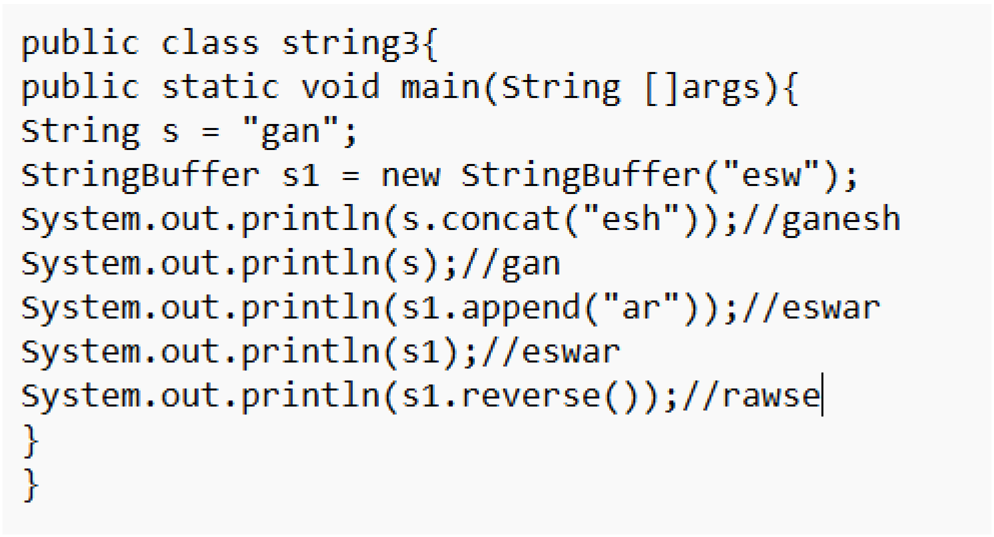
-

- In String Builder and String Buffer class hashCode () and equals () are not overridden but toString () is overridden.



```
public class string3{
public static void main(String []args){
StringBuffer s = new StringBuffer("ganesh");
StringBuffer s1 = new StringBuffer("ganesh");
System.out.println(s);
System.out.println(s1);
System.out.println(s.hashCode());
System.out.println(s1.hashCode());
System.out.println(s.equals(s1));
}
}
//ganesh
//ganesh
//1237598030
//688766789
//false
```

-
- Concat () method and append () method both are used to perform concatenation.
- Concat () accept only String type and available in String class only.
- Append () is an overloaded method which accepts all the types.
- Reverse () is used to reverse the given String and it is available only in StringBuilder and StringBuffer class.



```
public class string3{
public static void main(String []args){
String s = "gan";
StringBuffer s1 = new StringBuffer("esw");
System.out.println(s.concat("esh"));//ganesh
System.out.println(s);//gan
System.out.println(s1.append("ar"));//eswar
System.out.println(s1);//eswar
System.out.println(s1.reverse());//rawse
}
}
```

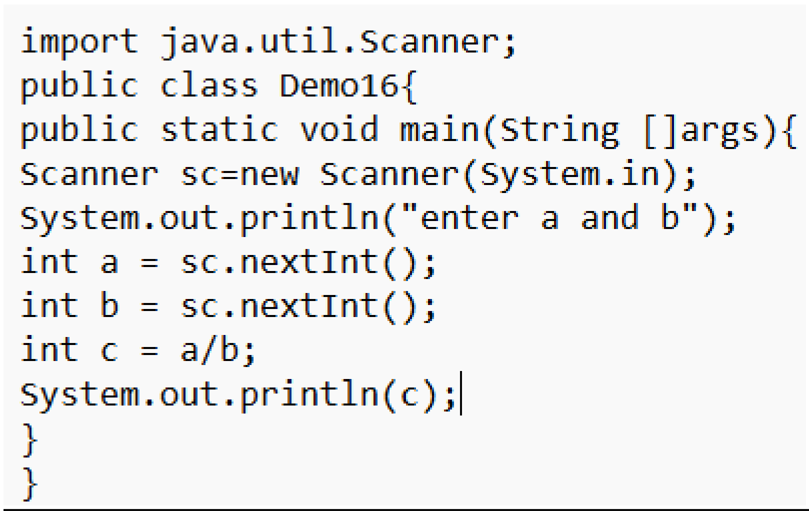
-
- Concat is only used to add something and just print, it doesn’t override or change the original String. It is only applicable to only Strings.
- Append () is used to change the original String and gives the final appended String as output and it accepts any type of input.

```
public class string3{
public static void main(String []args){
String s = "mom";
StringBuffer s1 = new StringBuffer(s);
System.out.println(s1);//mom
String s3 = new String(s1.reverse());
if(s.equals(s3)){
System.out.println("palindrome");
}
}
}
//palindrome
```

-
- Here the problem is we can use reverse method only for StringBuilder class or StringBuffer class and Equals method for only strings.
- So, we need to change string to buffer type to use reverse method and again that output of reverse method to string type to use equals method.

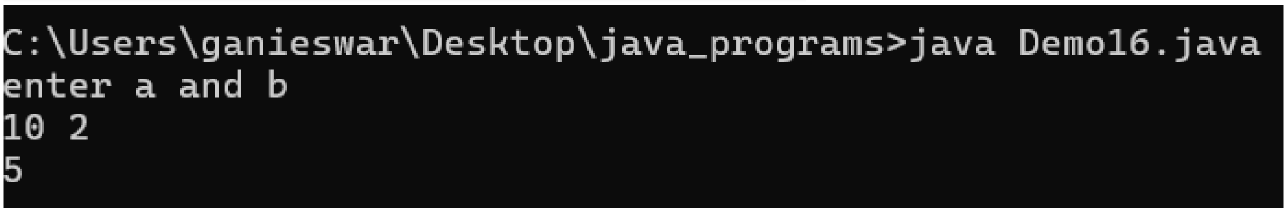
■ EXCEPTION HANDLING

- Exception is an unexpected event which terminates the flow of execution abnormally during the runtime by JVM.
- Exception in java is an error that occurs during program execution time and disturb the normal flow of execution.
- Termination will happen in three ways:
  Normal/ peaceful termination
  Forceful termination
  Abnormal termination
- **NORMAL TERMINATION**: A program executes without any error or exception. This termination is known as Normal/ peaceful termination.



```
import java.util.Scanner;
public class Demo16{
public static void main(String []args){
Scanner sc=new Scanner(System.in);
System.out.println("enter a and b");
int a = sc.nextInt();
int b = sc.nextInt();
int c = a/b;
System.out.println(c);
}
}
```

-



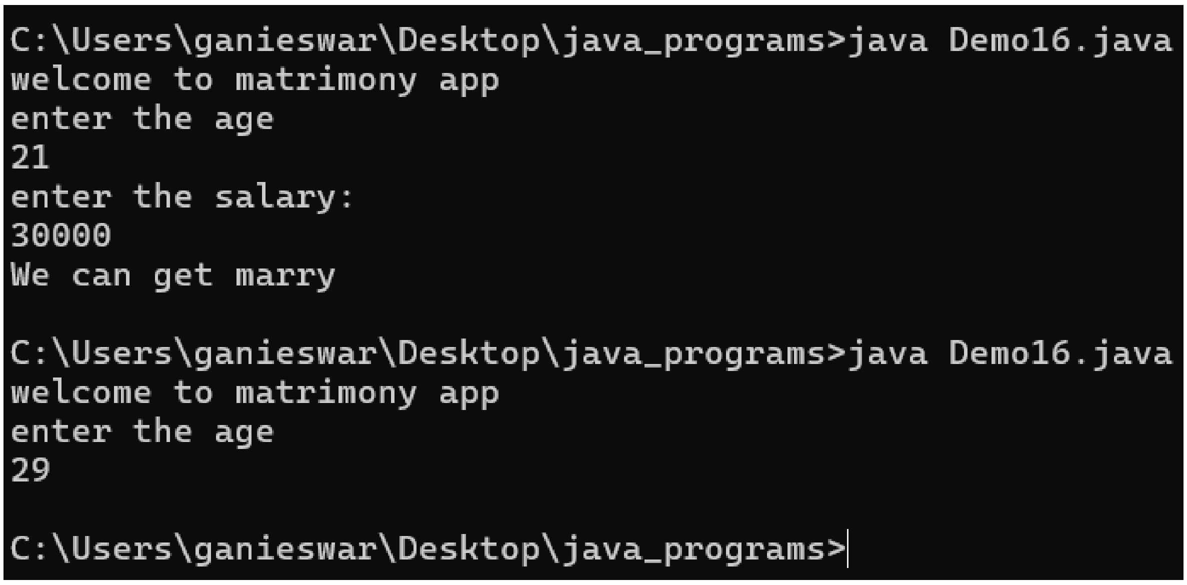
```
C:\Users\ganieswar\Desktop\java_programs>java Demo16.java
enter a and b
10 2
5
```

-

- **FORCEFUL TERMINATION:** The program termination done by programmer forcefully by using System.exit(0).

```
import java.util.Scanner;
public class Demo16{
public static void main(String []args){
System.out.println("welcome to matrimony app");
Scanner sc=new Scanner(System.in);
System.out.println("enter the age");
int age = sc.nextInt();
if(age>=18 && age<=25){
System.out.println("enter the salary:");
double salary = sc.nextDouble();
if(salary>=30000){
System.out.println("We can get marry");
}
else{
System.exit(0);
}
}
else{
System.exit(0);
}
}
}
```

-



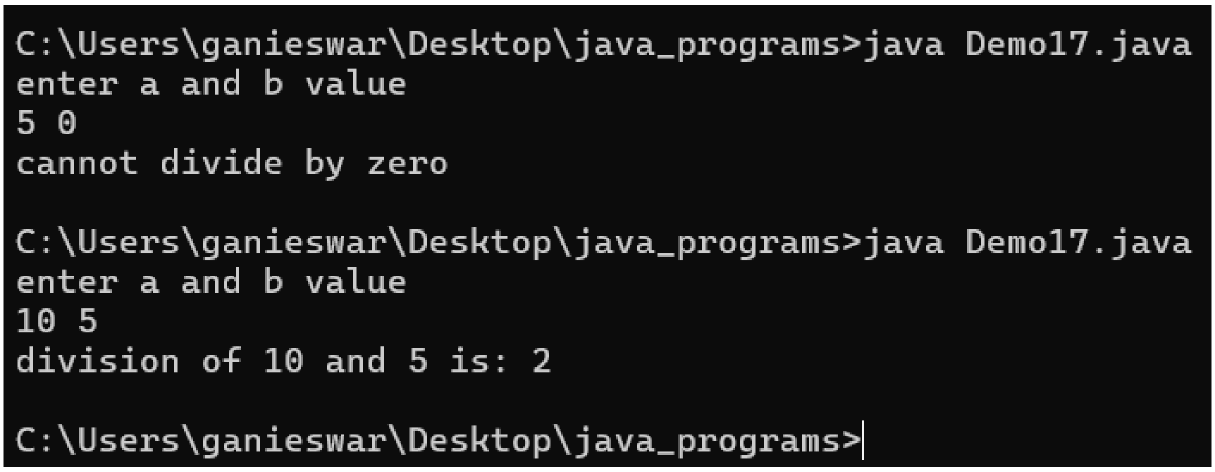
```
C:\Users\ganieswar\Desktop\java_programs>java Demo16.java
welcome to matrimony app
enter the age
21
enter the salary:
30000
We can get marry

C:\Users\ganieswar\Desktop\java_programs>java Demo16.java
welcome to matrimony app
enter the age
29

C:\Users\ganieswar\Desktop\java_programs>
```

-

- **ABNORMAL TERMINATION:** The program execution will be terminated unexpectedly/ abnormally by JVM during runtime.
- How to handle the exception: we can handle the exception by using try, catch, and finally block.
- Try block: try block is used to declare the critical statement/ dangerous statement. Try is a keyword.
- Catch Block: Catch block is used to handle the exception, without catch block we can execute the program but we cannot handle the exception.
- Finally Block: It is mainly used to execute an important statement.

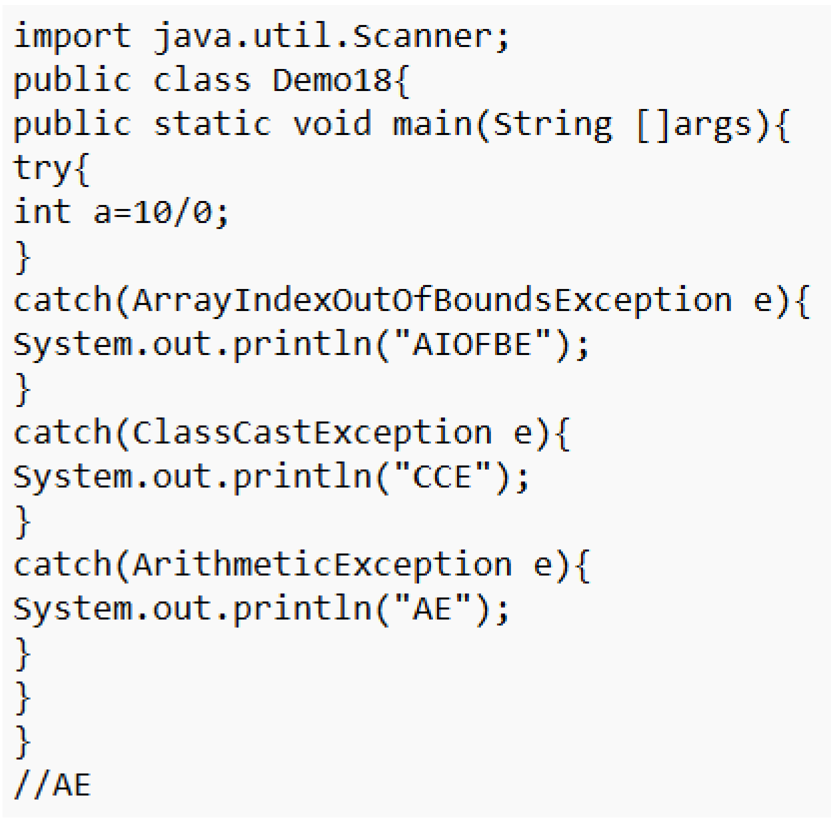
```
import java.util.Scanner;
public class Demo17{
public static void main(String []args){
Scanner sc=new Scanner(System.in);
System.out.println("enter a and b value");
int a = sc.nextInt();
int b = sc.nextInt();
try
{
int res=a/b;
System.out.println("division of "+a+" and "+b+" is:
"+res);
}
catch(Exception e){
System.out.println("cannot divide by zero");
}
finally{
sc.close();
}
}
}
```

-



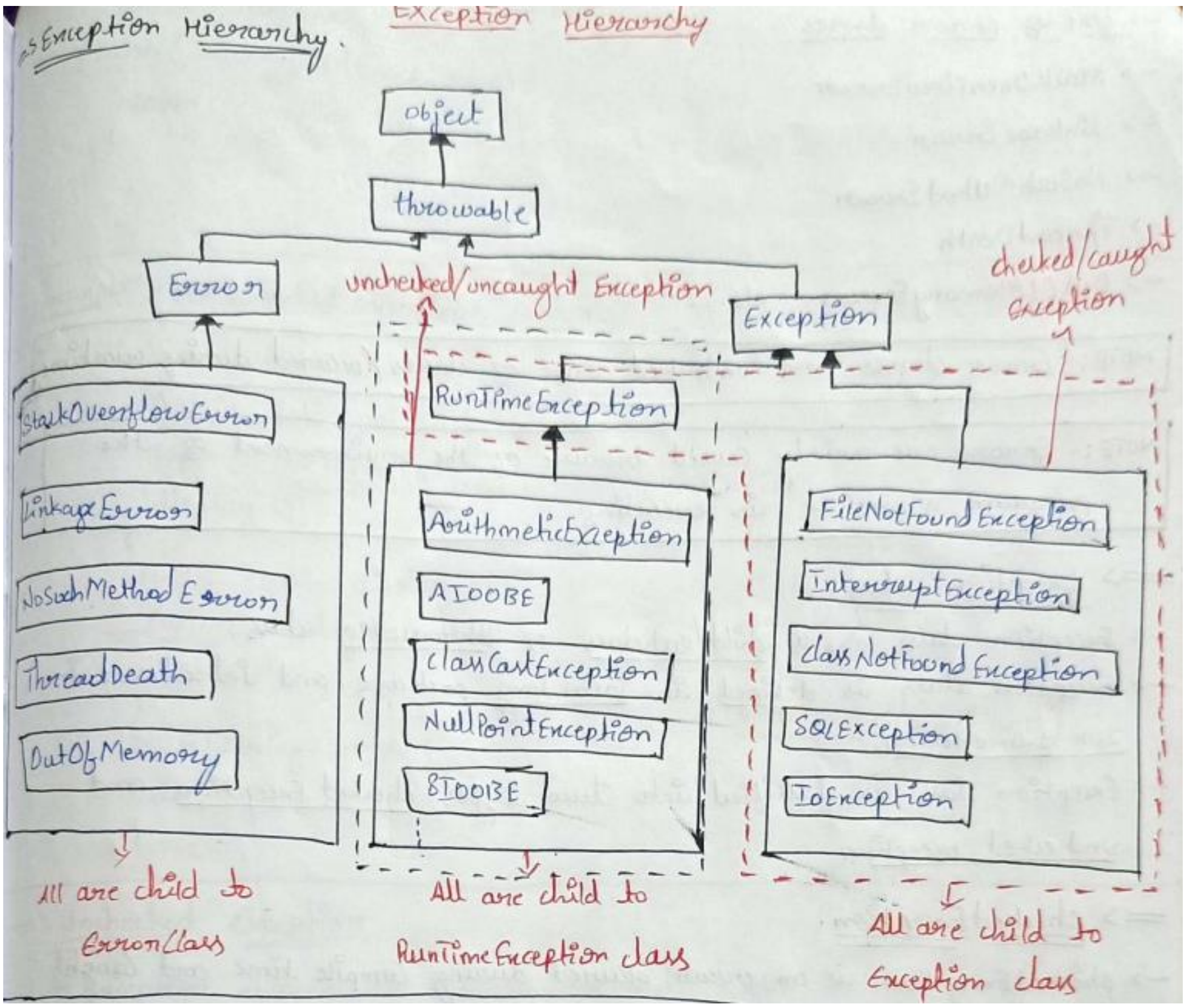
```
C:\Users\ganieswar\Desktop\java_programs>java Demo17.java
enter a and b value
5 0
cannot divide by zero

C:\Users\ganieswar\Desktop\java_programs>java Demo17.java
enter a and b value
10 5
division of 10 and 5 is: 2

C:\Users\ganieswar\Desktop\java_programs>
```

-

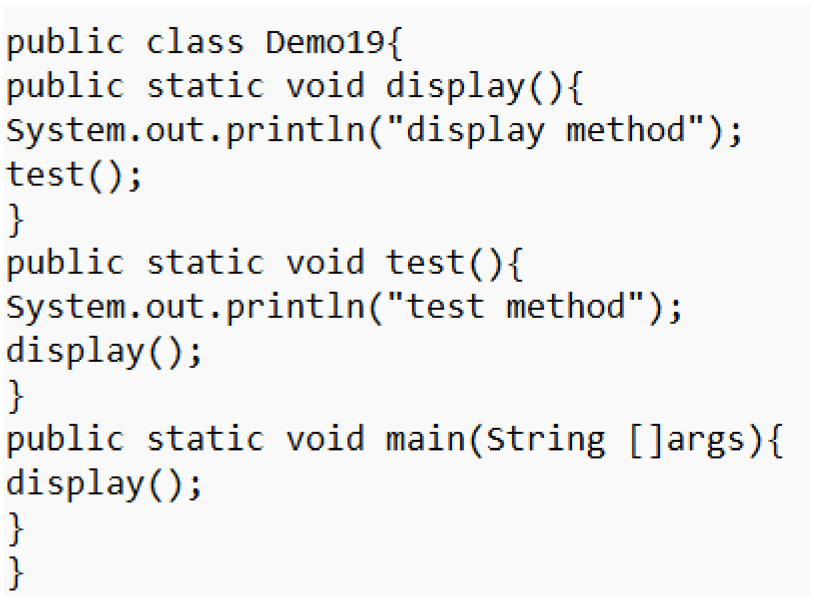
- We cannot declare try block without catch or finally block.
- To declare try block it is mandatory to declare catch or finally block.
- We cannot declare catch block without try block, try block is mandatory to be declared for catch block.
- We cannot declare finally block without try block, because finally block is dependent on try block.
- Finally block is not dependent on catch block because finally block will execute whether exception is handled or not.
- Without declaring catch block we cannot handle the exception even though finally block is declared.
- We cannot declare multiple try block for one catch block because catch blocks can accept only one try block.
- We can declare multiple catch blocks for a single try block (we must maintain child to parent hierarchy) but only one suitable catch block will get executed.
- 

```
import java.util.Scanner;
public class Demo18{
public static void main(String []args){
try{
int a=10/0;
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("AIOFBE");
}
catch(ClassCastException e){
System.out.println("CCE");
}
catch(ArithmeticException e){
System.out.println("AE");
}
}
}
//AE
```

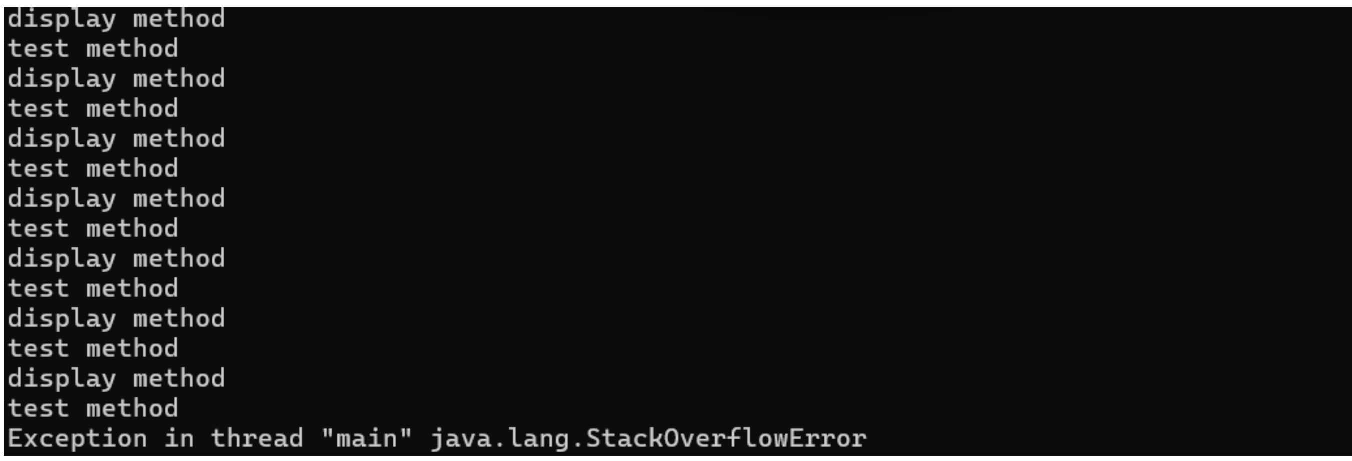
- We can declare only one finally block for one try block (can have multiple catch block) and finally block should be the last block because catch block is dependent on try block.



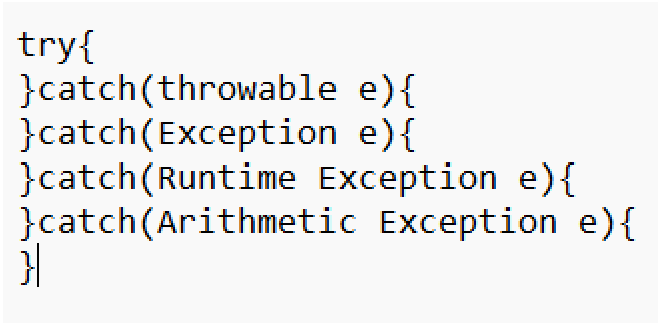
- **Throwable class** is the supermost class for all the error class and exception class. It is a child class of object class.
- It is defined in java.lang package.
- Throwable class instance (object creation) can be thrown by JVM or can be thrown by using throw statement.
- **Error class** is a child class of throwable class. It is defined in java.lang package
- We cannot try to catch the error classes. Some of the error classes are StackOverFlowError, LinkageError, NoSuchMethodError, ThreadDeath, OutOfMemoryError etc...
- Error classes are occurred during runtime, errors are mainly caused because of the environment of the program where it is executing.
- **Exception class** is a child of throwable class. It is defined in java.lang package
- It is classified into two types checked exception and unchecked exception.
- **Checked exception** is an error occurred during compile time and caught by compiler. These errors are mandatory to handle explicitly by the programmer else compiler will not allow the code to compile.

- Examples for checked exception are InterruptException, FileNotFoundException, SQLException, IOException.
- 

```
public class Demo19{
public static void display(){
System.out.println("display method");
test();
}
public static void test(){
System.out.println("test method");
display();
}
public static void main(String []args){
display();
}
}
```

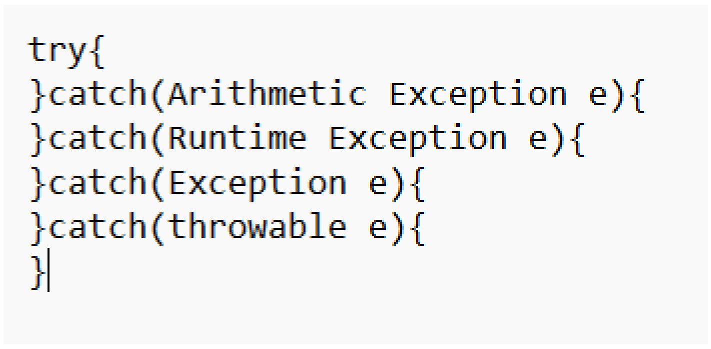
- 

```
display method
test method
display method
test method
display method
test method
display method
test method
display method
test method
display method
test method
display method
test method
Exception in thread "main" java.lang.StackOverflowError
```

- **Unchecked exception** escaped by the compiler and caught by the JVM during the runtime is known as Unchecked exception.
- Unchecked exceptions are mandatory to handle else JVM will terminate the program during runtime abnormally. (all runtime exceptions are unchecked exceptions)
- List of runtime Exceptions: ArithmeticException, ArrayIndexOutOfBoundEception, NullPointException, ClassCastException, StringIndexOutOfBoundException, etc…
- 

```
try{
}catch(throwable e){
}catch(Exception e){
}catch(Runtime Exception e){
}catch(Arithmetic Exception e){
}
```

- This is wrong way because the supermost class can handle all the errors and exceptions so child are not necessary.

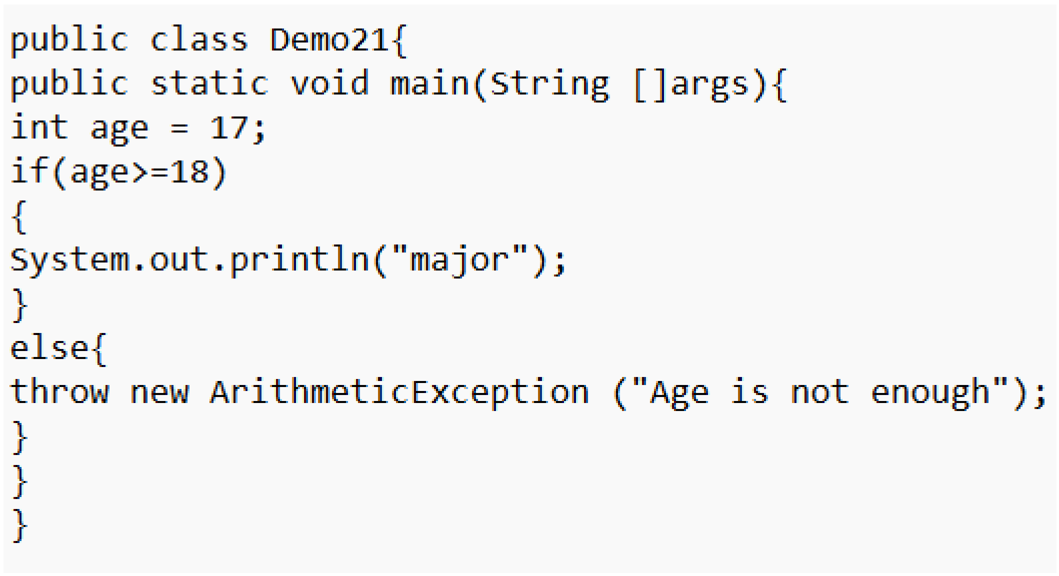


```
try{
}catch(Arithmetic Exception e){
}catch(Runtime Exception e){
}catch(Exception e){
}catch(throwable e){
}
```

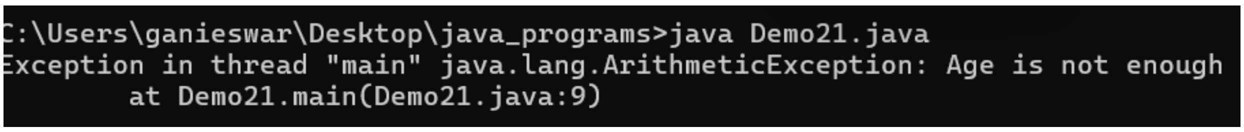
-
- This is correct way of hierarchy.
- Catch block hierarchy must be child to parent and it should not be parent to child.
- **THROWABLE STATEMENT:** These are used to throw an exception statement explicitly. There are two throwable statements. They are: 1. throw 2. throws
- **throw:**

  1.Throw is a control transfer statement. It is used to throw an exception explicitly.

  2.Throw statement should be the last statement and only one statement in a block.



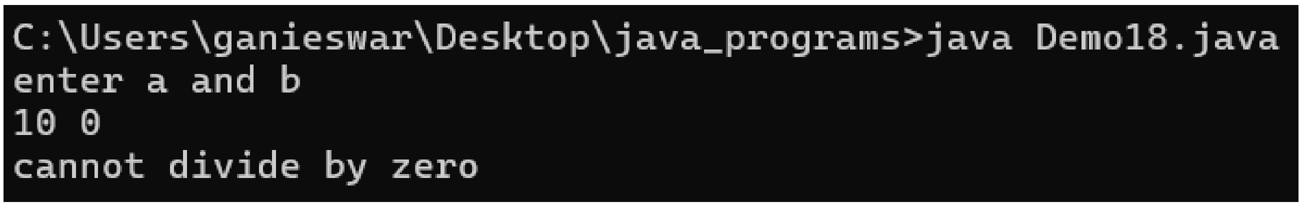
```
public class Demo21{
public static void main(String []args){
int age = 17;
if(age>=18)
{
System.out.println("major");
}
else{
throw new ArithmeticException ("Age is not enough");
}
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java Demo21.java
Exception in thread "main" java.lang.ArithmeticException: Age is not enough
        at Demo21.main(Demo21.java:9)
```

- **Throws:** throws is a keyword used to declare an exception to throw to the caller. It declares that a method might throw an exception. It can declare multiple Exceptions and must be declared in method signature/ declaration.

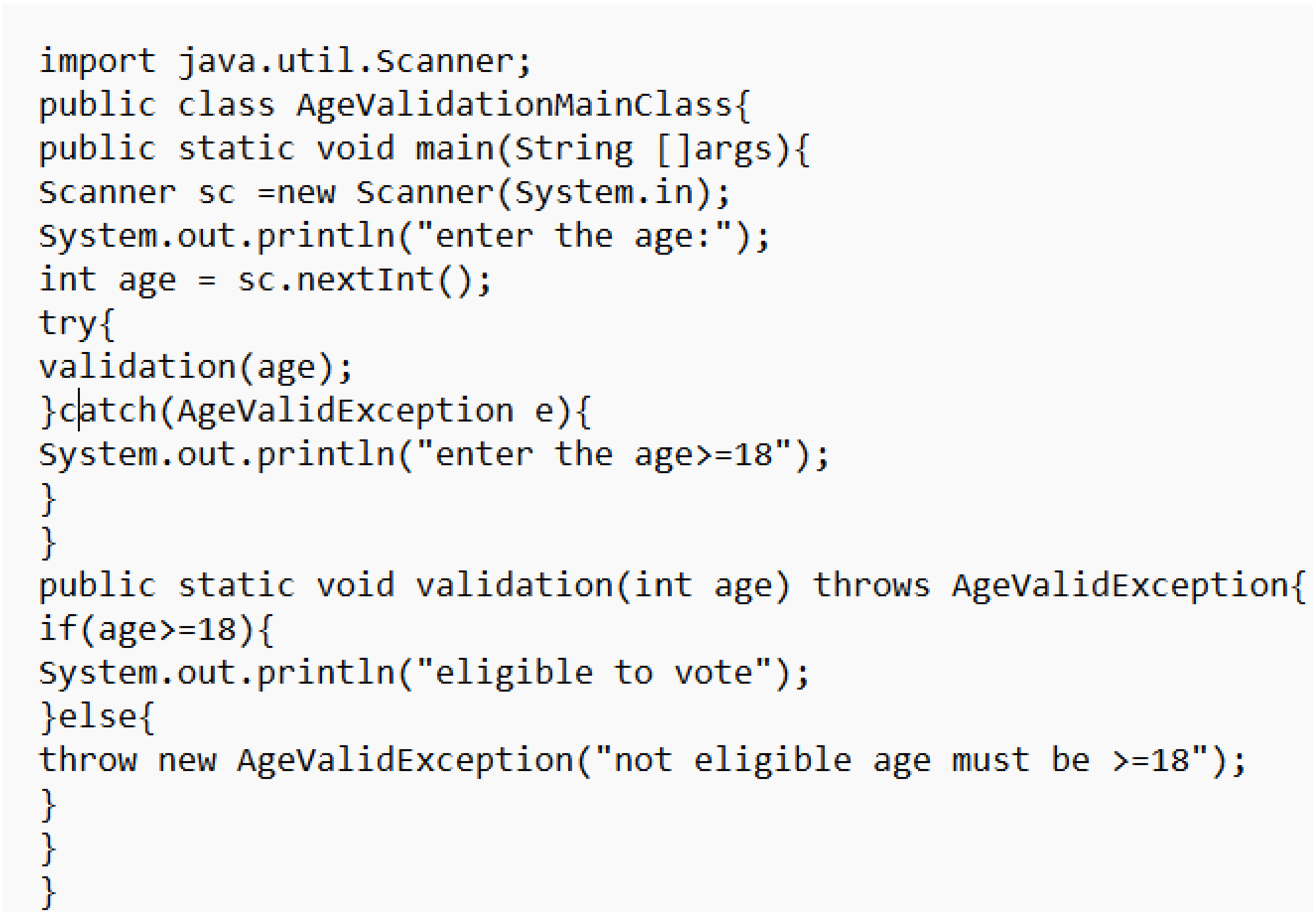
```java
import java.util.Scanner;
public class Demo18{
public static void div(int a, int b) throws ArithmeticException{
int res=a/b;
System.out.println(res);
}
public static void main(String []args){
Scanner sc = new Scanner(System.in);
System.out.println("enter a and b");
int a = sc.nextInt();
int b = sc.nextInt();
try{
div(a,b);
}catch(Exception e){
System.out.println("cannot divide by zero");
}
}
}
```



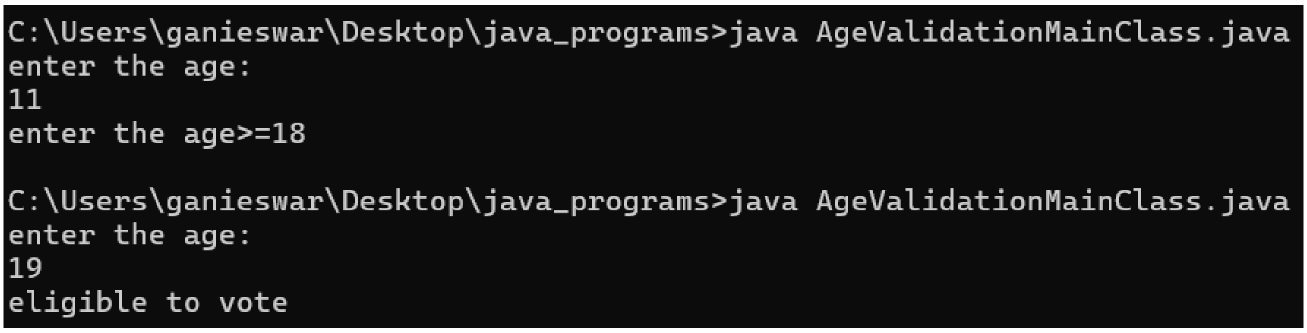
```
C:\Users\ganieswar\Desktop\java_programs>java Demo18.java
enter a and b
10 0
cannot divide by zero
```

- **CUSTOM or USER-DEFINED EXCEPTION**
- An exception which is created by programmer or developer explicitly is known as user-defined/ custom exception.
- Steps to create CustomException:
- Create a class and classname must end with Exception word.
- A created Exception class extends the Exception or throwable class.
- Design a constructor and invoke exception class constructor to initialize the reason of the exception.

```java
public class AgeValidException extends Exception{
public AgeValidException(String message){
super(message);
}
}
```

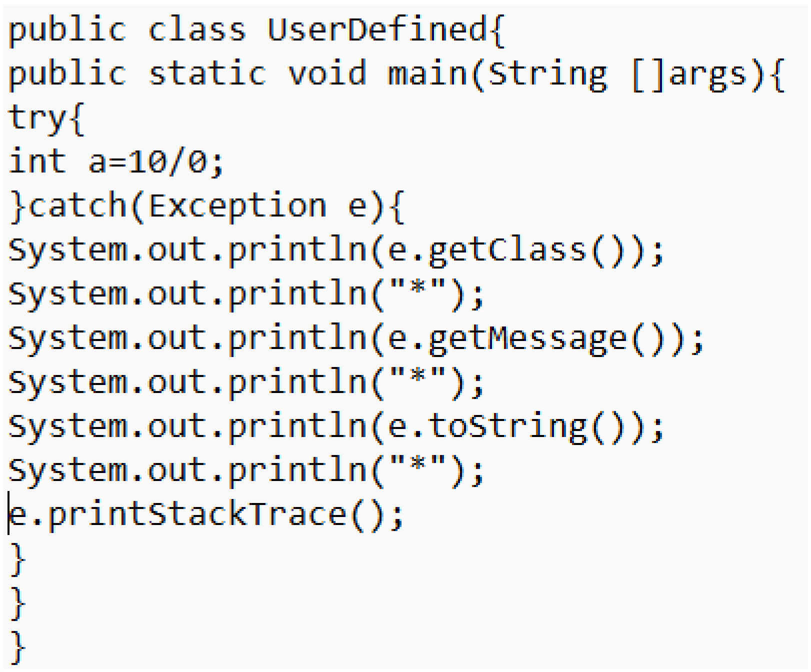
- 

```java
import java.util.Scanner;
public class AgeValidationMainClass{
public static void main(String []args){
Scanner sc =new Scanner(System.in);
System.out.println("enter the age:");
int age = sc.nextInt();
try{
validation(age);
}catch(AgeValidException e){
System.out.println("enter the age>=18");
}
}
public static void validation(int age) throws AgeValidException{
if(age>=18){
System.out.println("eligible to vote");
}else{
throw new AgeValidException("not eligible age must be >=18");
}
}
}
```

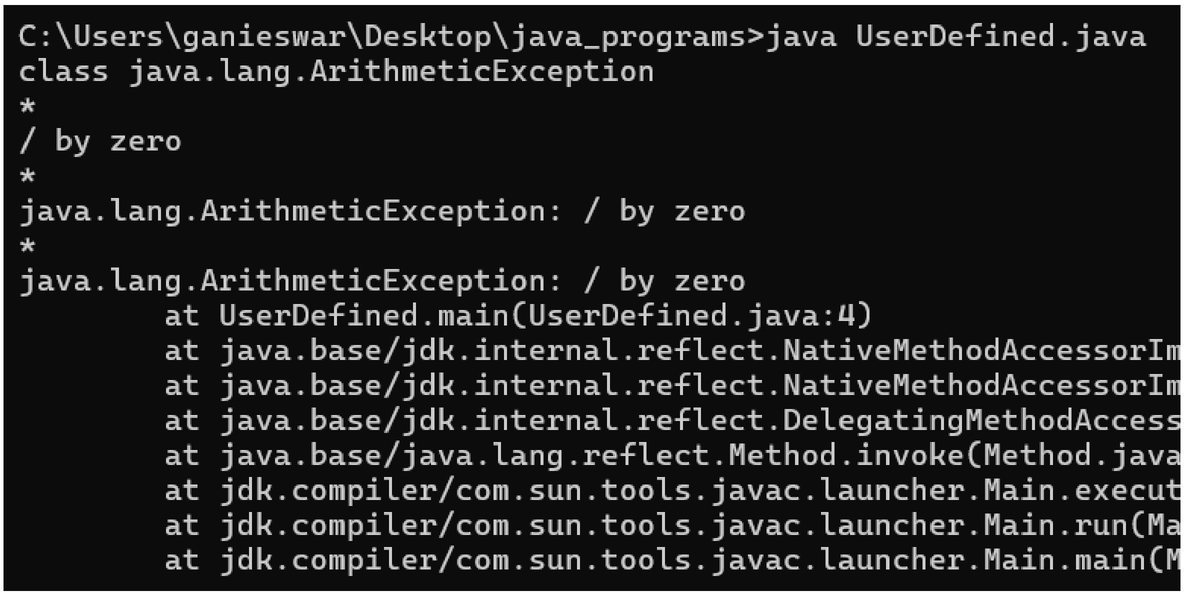
- 

```
C:\Users\ganieswar\Desktop\java_programs>java AgeValidationMainClass.java
enter the age:
11
enter the age>=18

C:\Users\ganieswar\Desktop\java_programs>java AgeValidationMainClass.java
enter the age:
19
eligible to vote
```

- Exception class pre-defined methods ():
- getMessage ()
  Return type is string and it defined in throwable class
  It is used to display the reason of the exception
- getClass ()
  return class like AE, AIOOBE, CCE etc…
  It is defined in object class.
  It is used to fetch the exception class which occurs.
- printStackTrace ()
  Return type is void and defined in throwable class
  It is used to print which exception occurs, why it occurs, line it occurs.



```
public class UserDefined{
public static void main(String []args){
try{
int a=10/0;
}catch(Exception e){
System.out.println(e.getClass());
System.out.println("*");
System.out.println(e.getMessage());
System.out.println("*");
System.out.println(e.toString());
System.out.println("*");
e.printStackTrace();
}
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java UserDefined.java
class java.lang.ArithmeticException
*
/ by zero
*
java.lang.ArithmeticException: / by zero
*
java.lang.ArithmeticException: / by zero
        at UserDefined.main(UserDefined.java:4)
        at java.base/jdk.internal.reflect.NativeMethodAccessorIm
        at java.base/jdk.internal.reflect.NativeMethodAccessorIm
        at java.base/jdk.internal.reflect.DelegatingMethodAccess
        at java.base/java.lang.reflect.Method.invoke(Method.java
        at jdk.compiler/com.sun.tools.javac.launcher.Main.execut
        at jdk.compiler/com.sun.tools.javac.launcher.Main.run(Ma
        at jdk.compiler/com.sun.tools.javac.launcher.Main.main(M
```

ARRAY

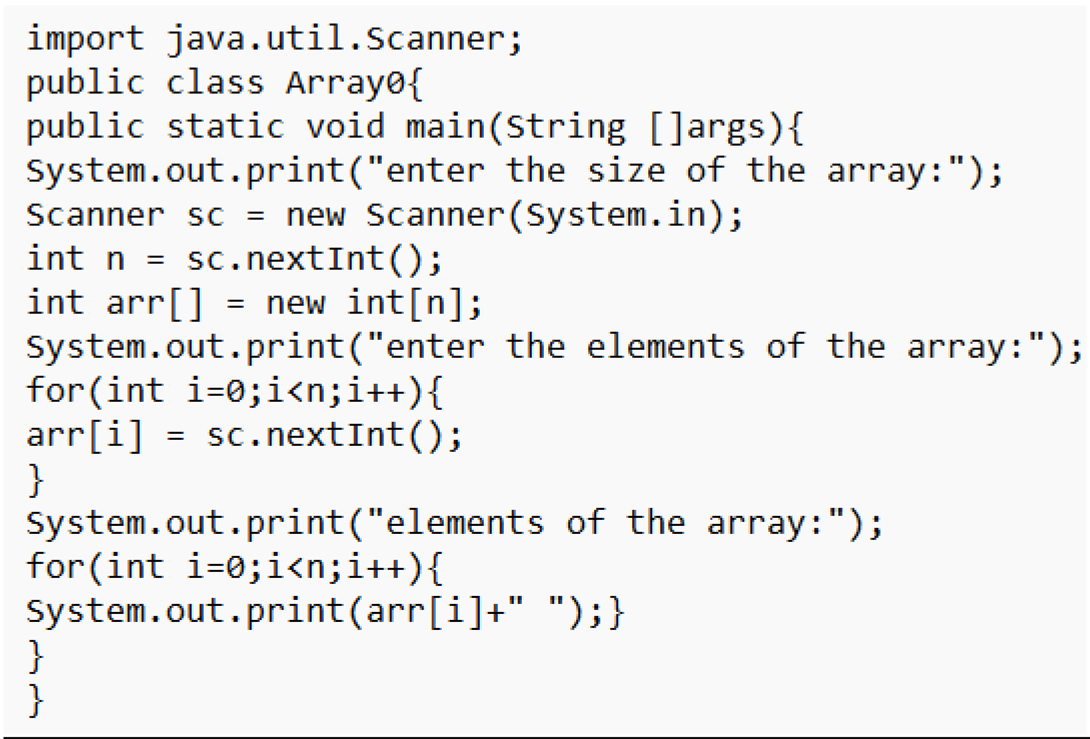
- Array is used to store the group of homogenous type of elements.
- Elements are stored based on index.
- Index will be start from zero and ends in (n-1) where n is length.
- Array is fixed in size/ length.
- If we try to store the element more than specified size then JVM will throw ArrayIndexOutOfBooundException during runtime.
- Array contains final variable that is length.
- datatype varname[] = new datatype[size];
  datatype[] varname = new datatype[size];
  varname[index] = value;

```
public class Array0{
public static void main(String []args){
int arr[] = new int[5];
arr[0] = 0;
arr[1] =1;
arr[2]=2;
arr[3]=3;
arr[4]=4;
for(int i=0;i<arr.length;i++){
System.out.print(arr[i]+" ");
}
}
}
//0 1 2 3 4
```

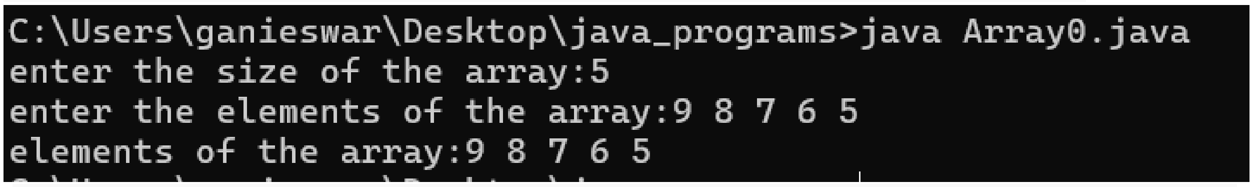
  Here we mention the size at the time of declaration and do initialization.
- datatype varname[] = {value1, value2, value3};
- datatype varname[] = new datatype[]{value1, value2, value3};

```
public class Array0{
public static void main(String []args){
int arr[] = {0,1,2,3,4};
for(int i=0;i<arr.length;i++){
System.out.print(arr[i]+" ");
}
}
}
//0 1 2 3 4
```

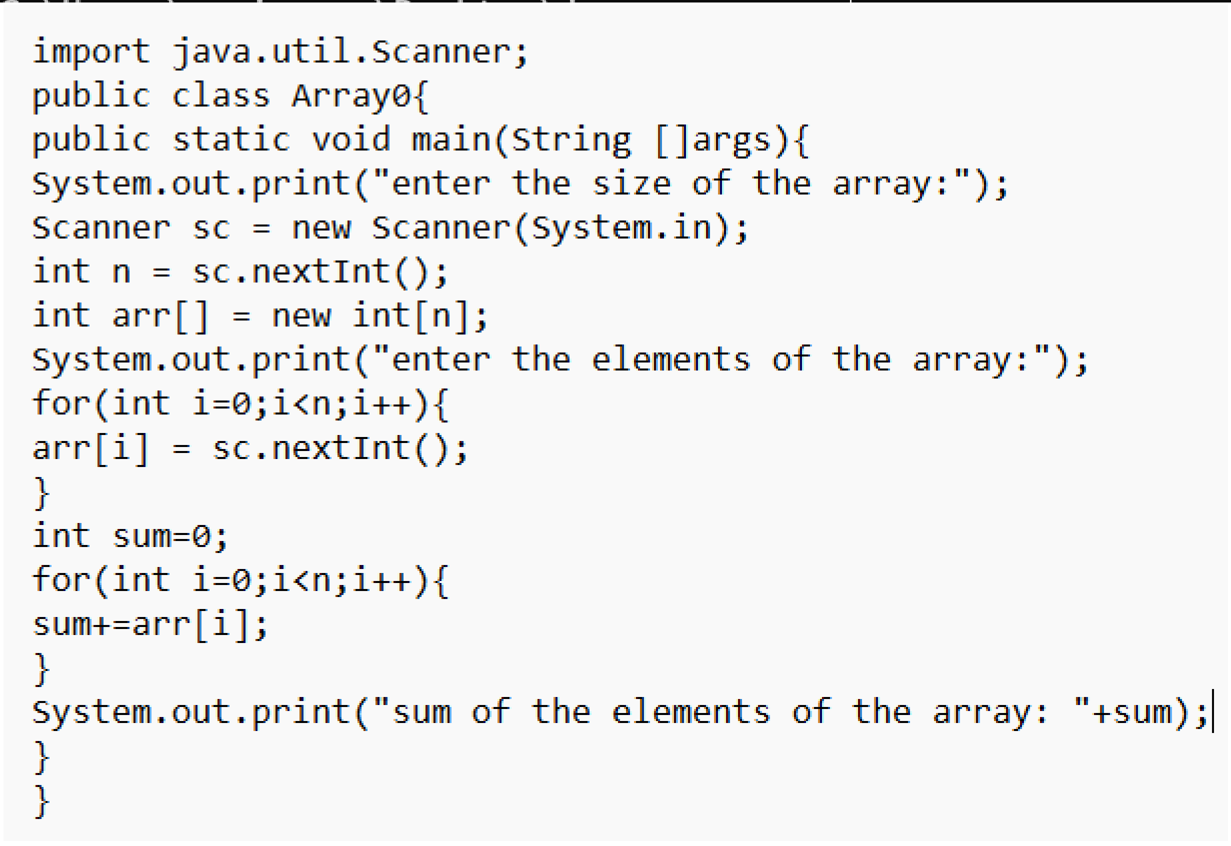
-
- taking input from the user:



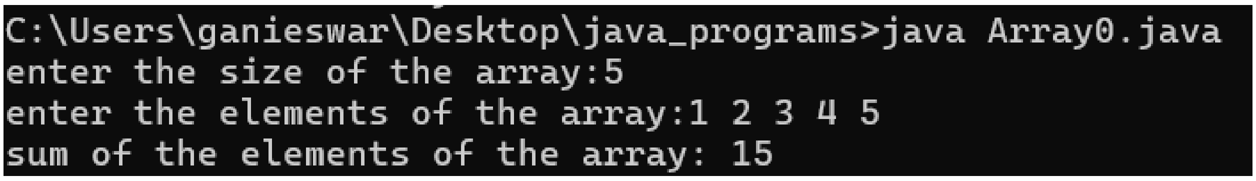
```
import java.util.Scanner;
public class Array0{
public static void main(String []args){
System.out.print("enter the size of the array:");
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int arr[] = new int[n];
System.out.print("enter the elements of the array:");
for(int i=0;i<n;i++){
arr[i] = sc.nextInt();
}
System.out.print("elements of the array:");
for(int i=0;i<n;i++){
System.out.print(arr[i]+" ");}
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java Array0.java
enter the size of the array:5
enter the elements of the array:9 8 7 6 5
elements of the array:9 8 7 6 5
```



```
import java.util.Scanner;
public class Array0{
public static void main(String []args){
System.out.print("enter the size of the array:");
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int arr[] = new int[n];
System.out.print("enter the elements of the array:");
for(int i=0;i<n;i++){
arr[i] = sc.nextInt();
}
int sum=0;
for(int i=0;i<n;i++){
sum+=arr[i];
}
System.out.print("sum of the elements of the array: "+sum);
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java Array0.java
enter the size of the array:5
enter the elements of the array:1 2 3 4 5
sum of the elements of the array: 15
```

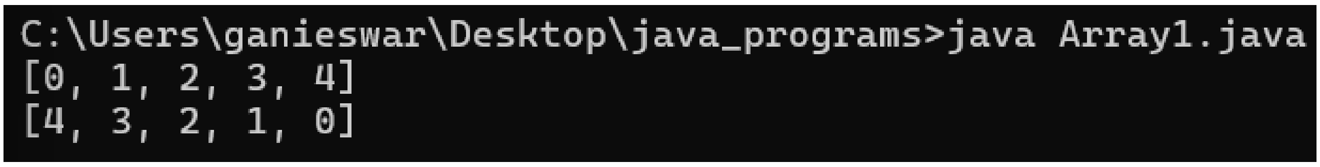
- Copying an array into other:

```
import java.util.Scanner;
public class Array0{
public static void main(String []args){
/*System.out.print("enter the size of the array:");
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
int arr[] = new int[n];
System.out.print("enter the elements of the array:");
for(int i=0;i<n;i++){
arr[i] = sc.nextInt();
}*/
int arr[]={0,1,2,3,4};
int arr1[] = new int[arr.length];
for(int i=0;i<arr1.length;i++){
arr1[i] = arr[i];
}
System.out.print("elements of the array1: ");
for(int i=0;i<arr1.length;i++){
System.out.print(arr1[i]+" ");
}
}
}
```

First create another array with same size and then copy all the elements.

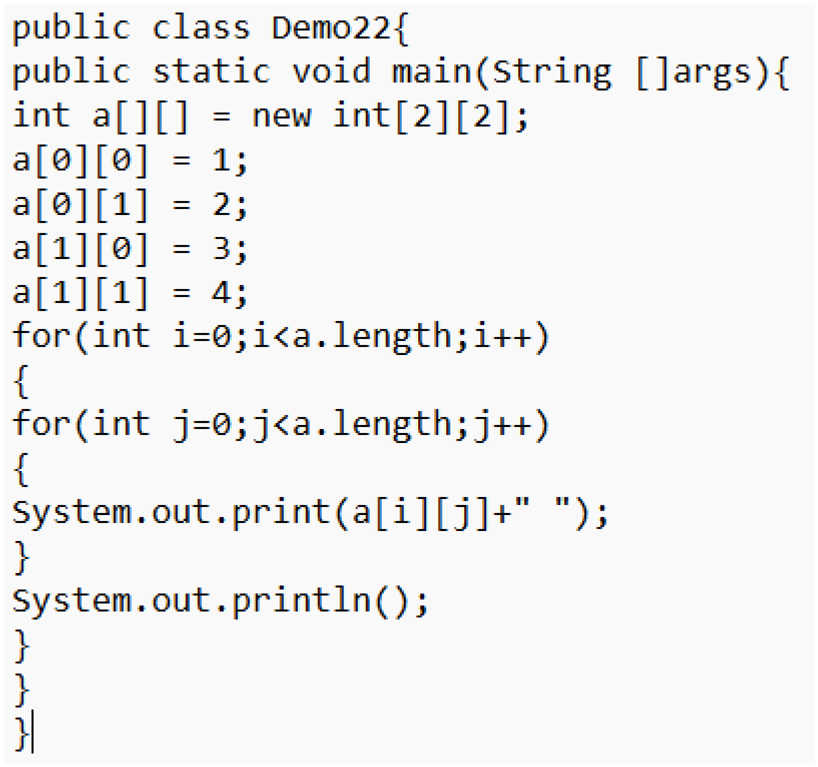
- Arrays.toString (argument) is used to print array elements.
- Reversing an array:

```
import java.util.Arrays;
public class Array1{
public static int[] reversedarray(int arr[]){
int reversed[] = new int[arr.length];
int j=0;
for(int i = (arr.length)-1;i>=0;i--){
reversed[j++] = arr[i];
}
return reversed;
}
public static void main(String []args){
int arr[] = {0,1,2,3,4};
System.out.println(Arrays.toString(arr));
int res[] = reversedarray(arr);
System.out.println(Arrays.toString(res));
}
}
```



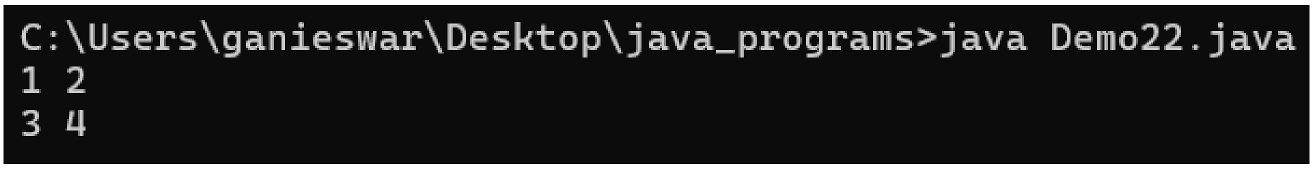
```
C:\Users\ganieswar\Desktop\java_programs>java Array1.java
[0, 1, 2, 3, 4]
[4, 3, 2, 1, 0]
```

- We need to import java.util.Arrays to use the Arrays.toString method.
- We can use this line instead of using variable j:
  reversed[((arr.length) -1)-i] = arr[i]; but it is easy to use j instead of this.
- Here we use integer array as a return type for the method.
- **MULTI-DIMENSINAL ARRAY:**
- datatype varname [][] = new datatype[row_size][column_size];
  datatype [][] varname = new datatype[row_size][column_size];
  varname [row_index][column_index] = value;
- datatype varname[][] = {{value1,value2},{value3,value4}};
  datatype varname = new datatype[][] {{v1,v2},{v3,v4}};



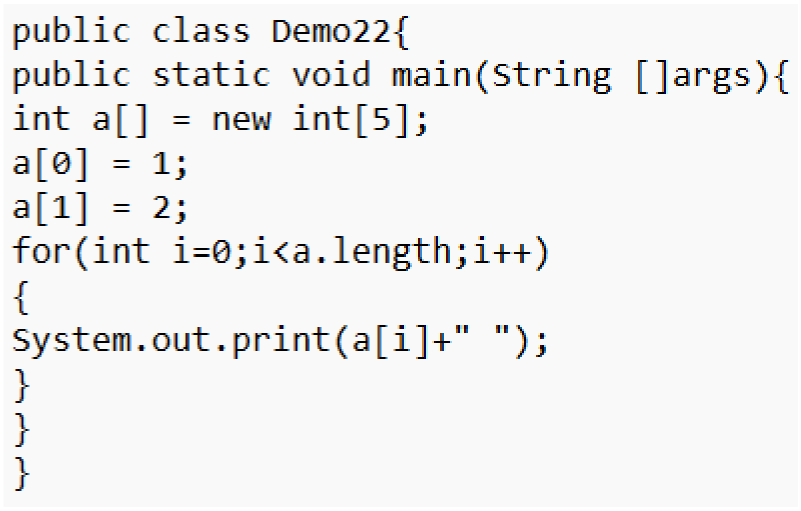
```
public class Demo22{
public static void main(String []args){
int a[][] = new int[2][2];
a[0][0] = 1;
a[0][1] = 2;
a[1][0] = 3;
a[1][1] = 4;
for(int i=0;i<a.length;i++)
{
for(int j=0;j<a.length;j++)
{
System.out.print(a[i][j]+" ");
}
System.out.println();
}
}
}
```

-

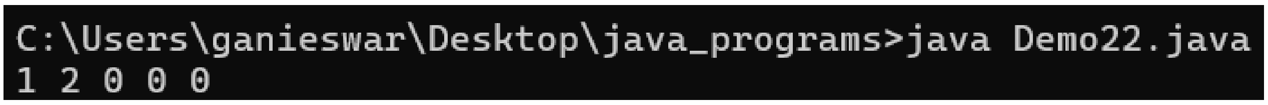


```
C:\Users\ganieswar\Desktop\java_programs>java Demo22.java
1 2
3 4
```

-
- Similarly, same for three-dimensional array.
- **Advantages** of arrays:
  It stores only homogeneous type of elements.
  It is fixed in size.
  Stores in sequential or based on index.
  By using array, we can perform sorting and searching
- **Disadvantages** of arrays:
  Arrays store only homogeneous and fixed in size.

- To overcome this problem collection concept got introduced.
- Collections can store homogenous as well as heterogeneous.
- Collections are not fixed in size, it is growable in nature.
- 

```
public class Demo22{
public static void main(String []args){
int a[] = new int[5];
a[0] = 1;
a[1] = 2;
for(int i=0;i<a.length;i++)
{
System.out.print(a[i]+" ");
}
}
}
```

- When we won't initialize memory arrays accept its default values based on the datatype.
- 

```
C:\Users\ganieswar\Desktop\java_programs>java Demo22.java
1 2 0 0 0
```

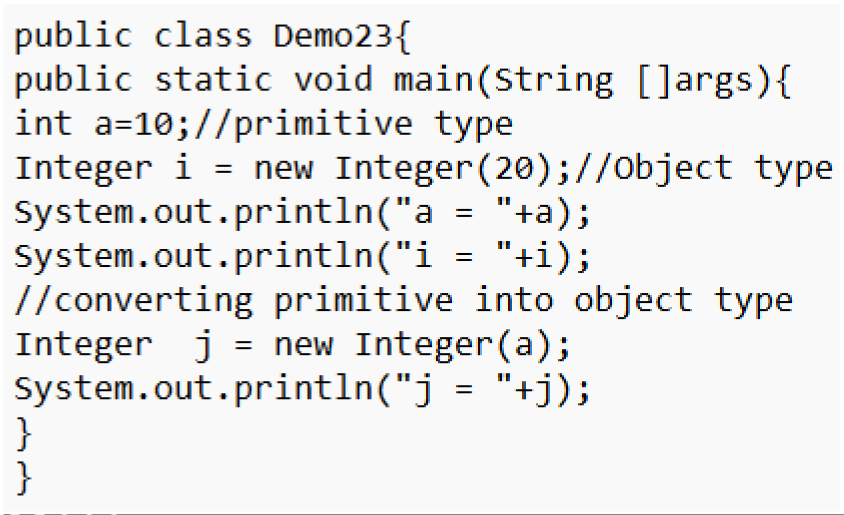
- Java is an Object-Oriented Language but not 100 % because of primitive datatypes as we cannot create object for primitive datatypes because primitive datatypes are fixed in size.
- Smalltalk and Ruby are 100% pure Object-Oriented programming language.
- To overcome this problem, we go for Wrapper class.

WRAPPER CLASS

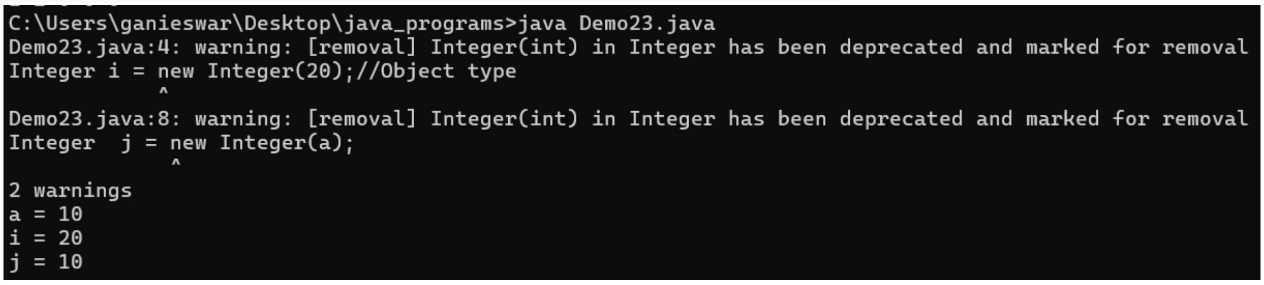
- It is a pre-defined class which is defines in java.lang package.
- It is a process of converting primitive datatype into object type (non-primitive datatype) and vice-versa (Object into primitive datatype).
- All wrapper class default value is NULL.

Primitive Datatype	Wrapper Class
byte	Byte
short	Short
int	Integer
double	Double
char	Character

- Wrapper class is classified into two types: Auto Boxing and Auto unboxing
- **Auto Boxing:**
- It is a process of converting primitive datatype into Object type implicitly.

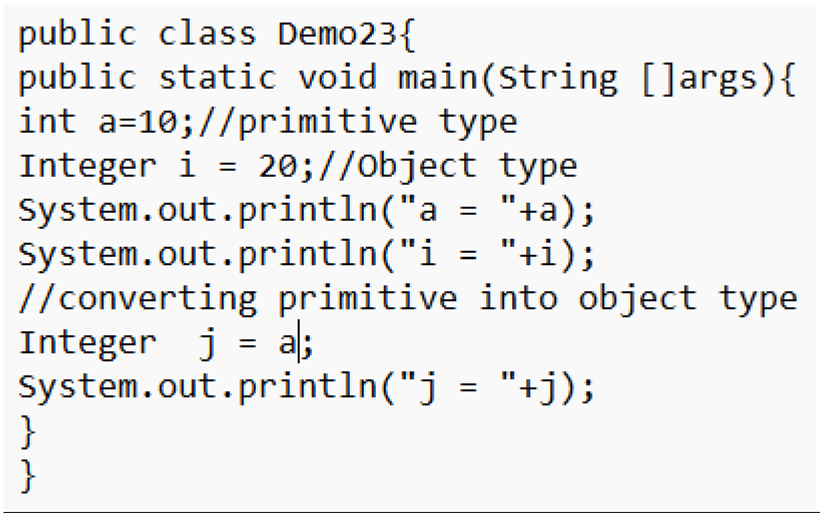


```java
public class Demo23{
public static void main(String []args){
int a=10;//primitive type
Integer i = new Integer(20);//Object type
System.out.println("a = "+a);
System.out.println("i = "+i);
//converting primitive into object type
Integer  j = new Integer(a);
System.out.println("j = "+j);
}
}
```

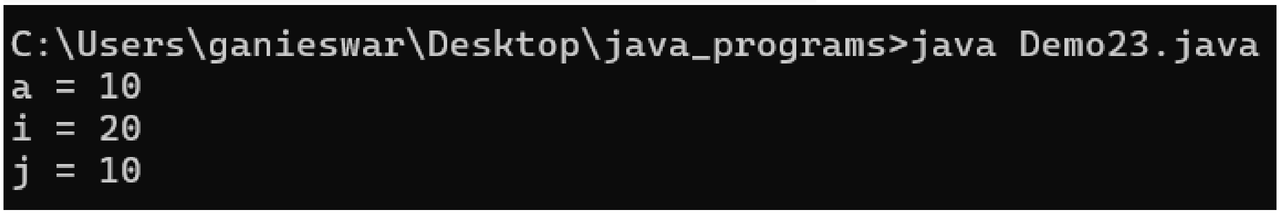
- 

```
C:\Users\ganieswar\Desktop\java_programs>java Demo23.java
Demo23.java:4: warning: [removal] Integer(int) in Integer has been deprecated and marked for removal
Integer i = new Integer(20);//Object type
            ^
Demo23.java:8: warning: [removal] Integer(int) in Integer has been deprecated and marked for removal
Integer  j = new Integer(a);
             ^
2 warnings
a = 10
i = 20
j = 10
```

- Here, as mentioned, using new Integer(a) is deprecated in Java 9 and later. A better approach would be:
  Integer j = a; // Auto-boxing

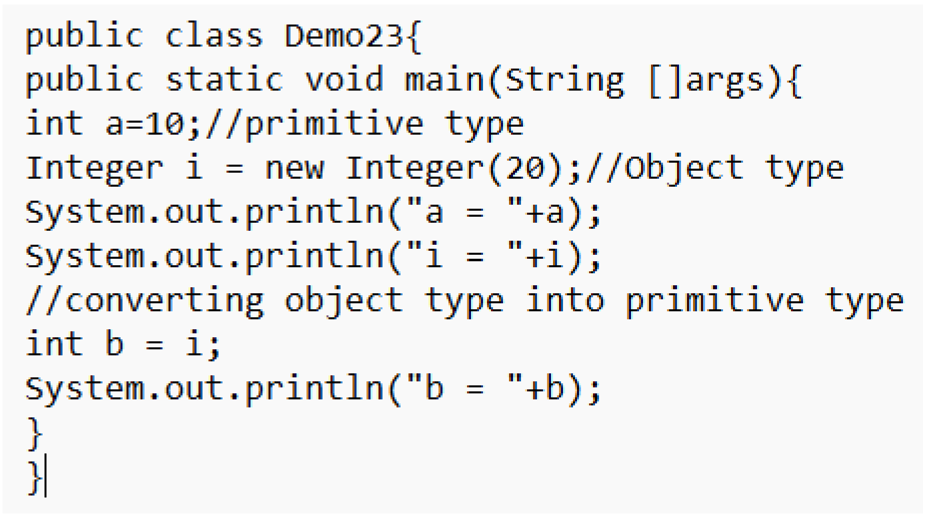


```java
public class Demo23{
public static void main(String []args){
int a=10;//primitive type
Integer i = 20;//Object type
System.out.println("a = "+a);
System.out.println("i = "+i);
//converting primitive into object type
Integer  j = a;
System.out.println("j = "+j);
}
}
```

- 

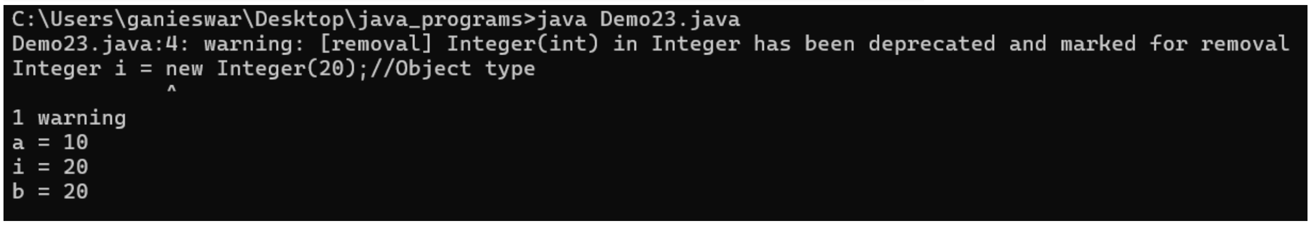
```
C:\Users\ganieswar\Desktop\java_programs>java Demo23.java
a = 10
i = 20
j = 10
```

- **AUTO UNBOXING:**
- It is a process of converting Object type into primitive type.



```
public class Demo23{
public static void main(String []args){
int a=10;//primitive type
Integer i = new Integer(20);//Object type
System.out.println("a = "+a);
System.out.println("i = "+i);
//converting object type into primitive type
int b = i;
System.out.println("b = "+b);
}
}
```

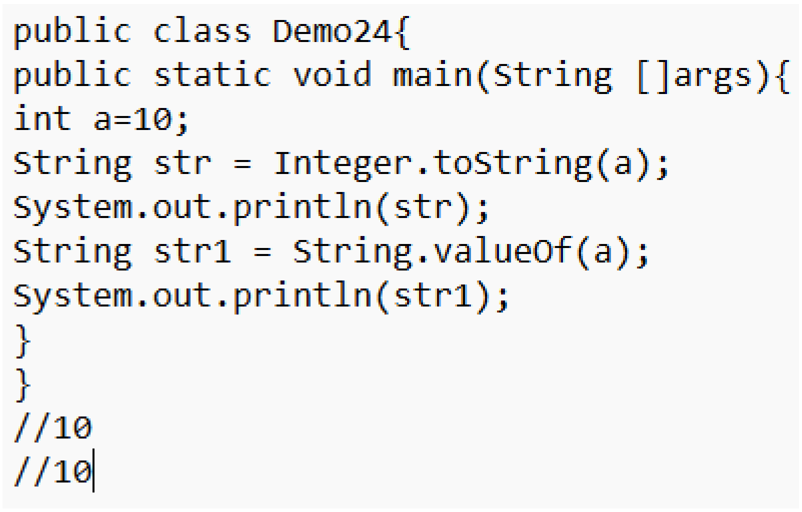
-



```
C:\Users\ganieswar\Desktop\java_programs>java Demo23.java
Demo23.java:4: warning: [removal] Integer(int) in Integer has been deprecated and marked for removal
Integer i = new Integer(20);//Object type
            ^
1 warning
a = 10
i = 20
b = 20
```

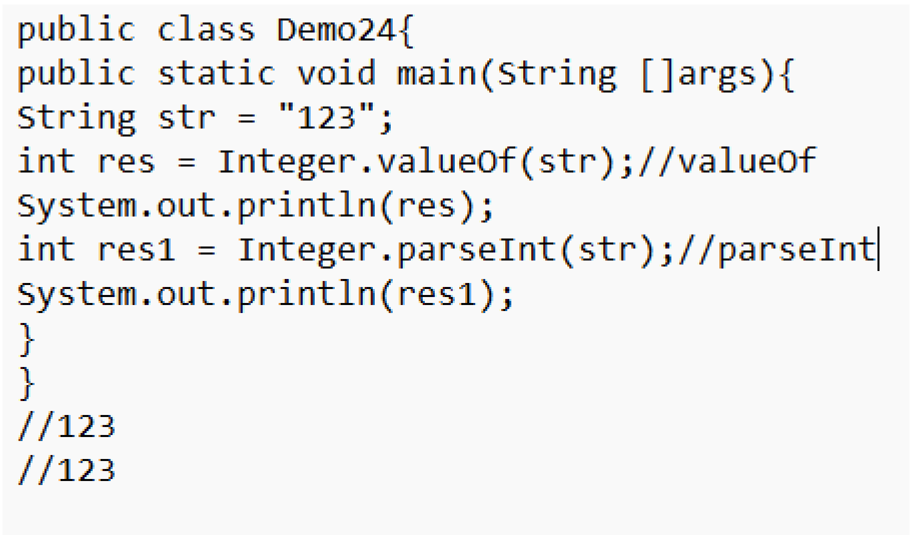
-

- Converting int into String:



```
public class Demo24{
public static void main(String []args){
int a=10;
String str = Integer.toString(a);
System.out.println(str);
String str1 = String.valueOf(a);
System.out.println(str1);
}
}
//10
//10
```

- Converting string to int:
  String value should be in number format to convert else JVM will throw an exception called number format exception. Like “123” is ok but “java “is not accepted.



```
public class Demo24{
public static void main(String []args){
String str = "123";
int res = Integer.valueOf(str);//valueOf
System.out.println(res);
int res1 = Integer.parseInt(str);//parseInt
System.out.println(res1);
}
}
//123
//123
```

- Palindrome or not:
  Here we need to use reverse method so, we need to change the int type to string type and reverse it and again convert to int to check with the original one. And one more important point is reverse method is in StringBuffer type.

```
public class Demo24{
public static int palindrome(int num){
StringBuffer str = new StringBuffer((Integer.toString(num)));
str.reverse();
return Integer.parseInt(str.toString());
}
public static void main(String []args){
int num = 121;
if(palindrome(num)==num){
System.out.println("palindrome");
}else{
System.out.println("not a palindrome");
}
}
}
//palindrome
```

  Here we need to change string buffer type to string type at return statement because Interger.parseInt () will accept only string type.

```
public static int palindrome(int num){
StringBuffer str = new
StringBuffer((Integer.toString(num)).reverse().toString);
return Integer.parseInt(str);
}
```

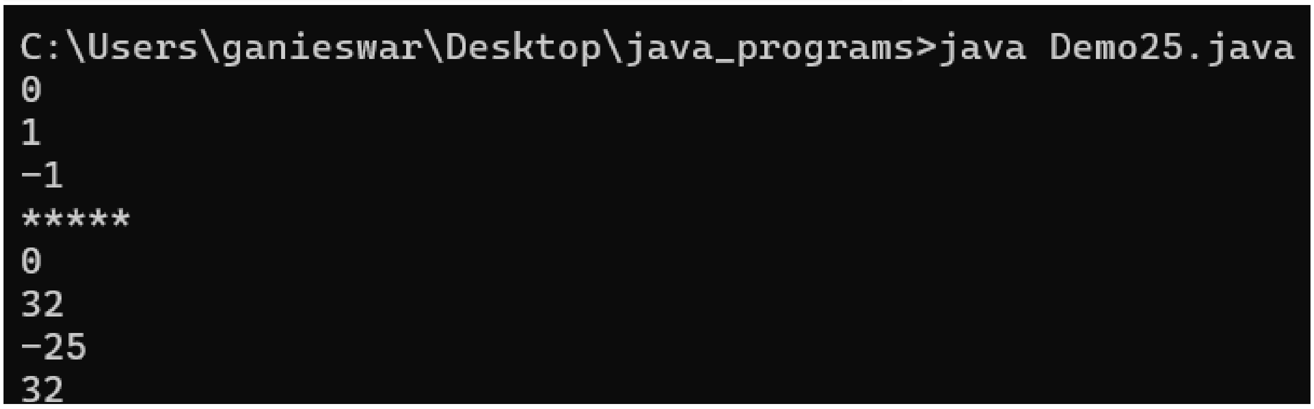
  If we try to directly do it one step, here Integer.toSting (num) gives us String type but reverse method in in StringBuffer or StringBuilder class, so we need to perform reverse on their objects.

- Converting Decimal to Binary and vice-versa:

```
public class Demo25{
public static void main(String []args){
int a=10;
//converting int to binary
System.out.println(Integer.toBinaryString(a));
String s = "1001";
//converting binary to int
System.out.println(Integer.parseInt(s,2));
}
}
//1010
//9
```

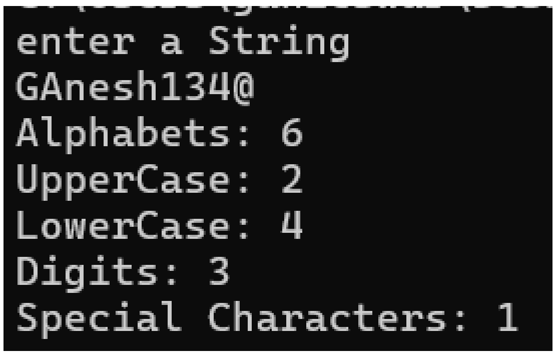
- **COMPARE:**
- Integer.compare (int a, int b) is used to compare int values and its return type is int.
  String1.compareTo(String2) is used to compare strings and its return type is also int.
  If the output is zero – equal
  Negative – b>a
  Positive – a>b

```
public class Demo25{
public static void main(String []args){
System.out.println(Integer.compare(10,10));
System.out.println(Integer.compare(20,10));
System.out.println(Integer.compare(10,20));
System.out.println("*****");
String name = "a";
String name1 = "A";
System.out.println(name.compareTo("a"));
System.out.println(name.compareTo("A"));
System.out.println(name.compareTo("z"));
System.out.println(name.compareTo(name1));
}
}
```



```
C:\Users\ganieswar\Desktop\java_programs>java Demo25.java
0
1
-1
*****
0
32
-25
32
```

- **CHARACTER CLASS:**
- Find the count of alphabets, uppercase, lowercase, digits and special characters in gives string.



```
enter a String
GAnesh134@
Alphabets: 6
UpperCase: 2
LowerCase: 4
Digits: 3
Special Characters: 1
```

-

```java
import java.util.Scanner;
public class Demo26{
public static void countAll(String str){
int count,count1,count2,count3,count4;
count=count1=count2=count3=count4=0;
for(int i=0;i<str.length();i++){
char ch = str.charAt(i);
if(Character.isAlphabetic(ch)){
count++;
if(Character.isUpperCase(ch)){
count1++;
}
else{
count2++;
}
}
else if(Character.isDigit(ch)){
count3++;
}
else{
count4++;
}
}
System.out.println("Alphabets: "+count);
System.out.println("UpperCase: "+count1);
System.out.println("LowerCase: "+count2);
System.out.println("Digits: "+count3);
System.out.println("Special Characters: "+count4);
}

public static void main(String []args){
Scanner sc= new Scanner(System.in);
System.out.println("enter a String");
String str = sc.nextLine();
countAll(str);
}
}
```