CORE JAVA

PANKAJ SIR ACADEMY NOTES

BATCH
20th APRIL 2022

Prepared by:

_M$A_


PA
QUALITY ASSURED

INDEX

_M$A_

21/04/2022 (Thursday)

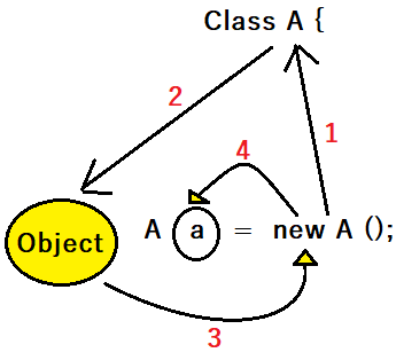
Syntaxes: { }, ( ), ;

Keywords in JAVA

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

NEW (Keyword):

NEW Keyword Sends request to the class, & class creates Object. Once the Object is created, NEW keyword will get Object's address in a reference variable.



GARBAGE COLLECTOR:

Garbage Collector on regular basis keeps removing unused objects from RAM, thus avoiding overflow of memory, this helps us manage memory efficiently.

22/04/2022 (Friday)

NON-STATIC/INSTANCE VARIABLE:

These variables are created inside class but outside method without using "static" keyword. Without creating object this variable cannot be accessed & hence we get error as shown below:

A.java

```
1
2 public class A {
3         int x = 10;
4     public static void main(String[] args) {
5         System.out.println(x);
6
7
8 }
9 }
10
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 10, 2022, 5:21:04 PM – 5:21:04 PM) [pid: 12996]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot make a static reference to the non-static field x

        at A.main(A.java:5)
```


Let's correct it:

A.java

```
1
2 public class A {
3         int x = 10;
4     public static void main(String[] args) {
5         A a1 = new A(); //Created new object.
6         System.out.println(a1.x);
7
8
9 }
10 }
11
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 10, 2022, 5:22:24 PM – 5:22:24 PM) [pid: 7500]
10
```

Every time we create an object a copy of non-static member will be loaded into that object.

```
1
2  public class B {
3          int  x = 10; //It's Non-Static & It gets loaded into object (Main Copy)
4          static int y = 20; //It's Static & gets loaded into Meta Space.
5      public static void main(String[] args) {
6          B a1 = new B();
7          a1.x = 100; //value changed, as it's the xerox of main copy it won't effect main copy
8          B a2 = new B();
9          B a3 = new B();
10         System.out.println(a1.x);
11         System.out.println(a2.x);
12         System.out.println(a3.x);
13         System.out.println(B.y); //Because It's static, we're calling it with ClassName w/o creating object
14     }
15 }
```

int x = 10; int x = 10; 100; int x = 10;

Markers Properties Servers Data Source Explorer Snippets Console

<terminated> B [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 10, 2022, 9:53:53 PM – 9:53:54 PM) [pid: 2568]

```
100
10
10
20
```

STATIC VARIABLE:

These variables are created inside the class & outside method using "static" keyword, these variables belong to class and can be accessed with class name. .i.e

B.java C.java

```
1
2  public class C {
3      static int x =10;
4      public static void main(String[] args) {
5          System.out.println(C.x);
6      }
7  }
8
```

Markers Properties Servers Data Source Explorer Snippets Console

<terminated> C [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 10, 2022, 11:09:08 PM – 11:09:08 PM) [pid: 10724]

```
10
```

25/04/2022 (Monday)

STATIC VARIABLE: (Contd...)

The value of static variables can be changed as shown in the below Ex:

A.java

```
1
2 public class A {
3     static int x = 10;
4     public static void main(String[] args) {
5         System.out.println(A.x);
6         A.x = 25;
7         System.out.println(A.x);
8     }
9 }
10
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (4) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 10, 2022, 11:19:04 PM – 11:19:04 PM) [pid: 14272]
10
25
```

METHODS:

Methods can be Static or Non Static, If we write "static" in method then it belongs to Object, If "static" is not written then it is Non Static.

Ex: `public static void main(String[] args)` (Static Method)
`public void test() {` (non static method)

To call a non static method, we need to create an object then only we can call it. i.e:

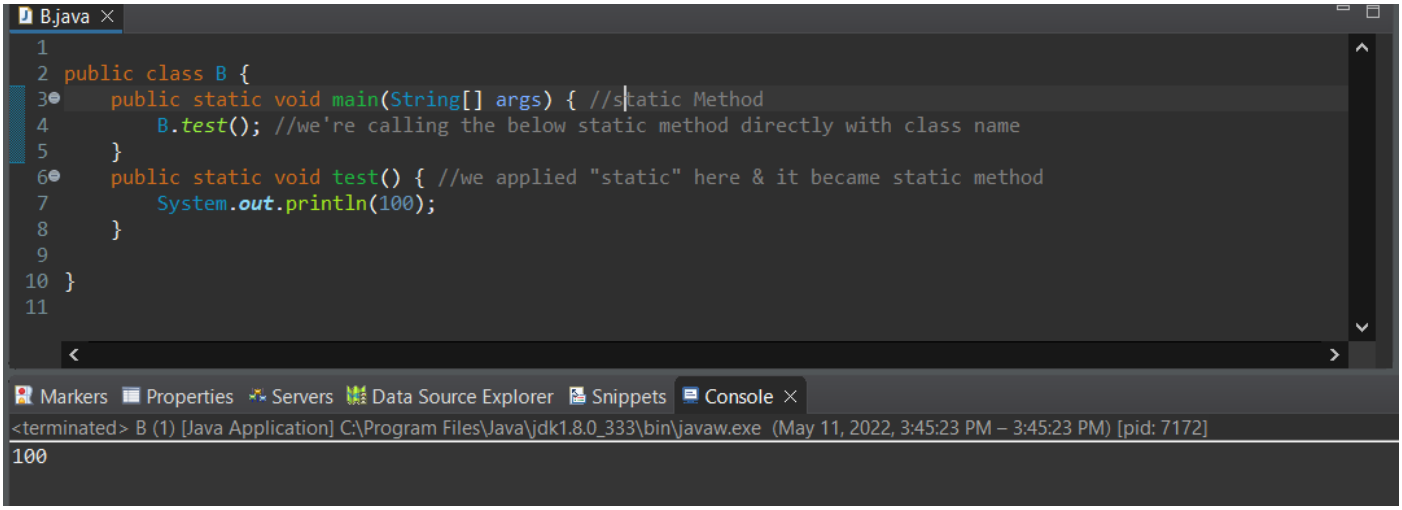
B.java

```
1
2 public class B {
3     public static void main(String[] args) { //Static Method
4         B a1 = new B();
5         a1.test(); //we are calling the below non static method
6     }
7     public void test() { //non static method
8         System.out.println(100);
9     }
10
11 }
12
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> B (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 11, 2022, 3:28:45 PM – 3:28:45 PM) [pid: 9516]
100
```

If we apply "static" to 2nd method (in above example) then it becomes static method, for static methods no need to create an object for calling, we can call it directly with class name, see the next Ex.

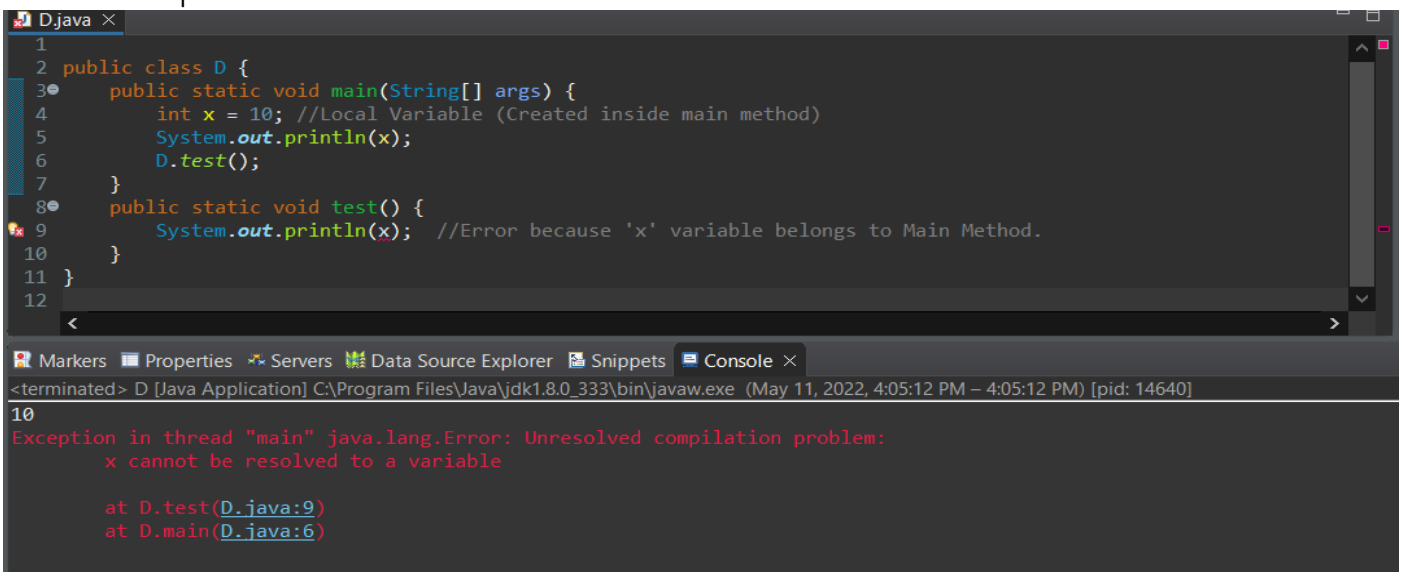


Types of variables in JAVA

The following are the types of variables in JAVA

1) Local Variable
2) Static Variable
3) Non-Static Variable
4) Reference Variable

1. Local Variable: These variables are created inside a method & should be used only within created method. If used outside created method, then we'll get an error as shown in the example below.



Ex: 2

```
1
2 public class D {
3     public static void main(String[] args) {
4         D.test();
5     }
6     public static void test() {
7         int x = 10; //Local Variable (Created inside test method)
8         System.out.println(x);
9     }
10 }
11
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> D [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 11, 2
10
```

Correct

```
1
2 public class D {
3     public static void main(String[] args) {
4         D.test();
5         System.out.println(x); //Error because x belongs to Test Method
6     }
7     public static void test() {
8         int x = 10; //Local Variable (Created inside test method)
9         System.out.println(x);
10    }
11 }
12
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> D [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 11, 2022, 4:19:47 PM
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        x cannot be resolved to a variable

        at D.main(D.java:5)
```

Error

```
1
2 public class D {
3     public static void main(String[] args) {
4         int x = 10;
5         System.out.println(x);  //Correct, because 'x' belongs to main method & used in main method
6     }
7 }
8
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> D [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 11, 2022, 4:24:19 PM – 4:24:19 PM) [pid: 14712]
10
```

Correct

It is mandatory to initialize local variable before using it or else we get an error as shown in the below example:

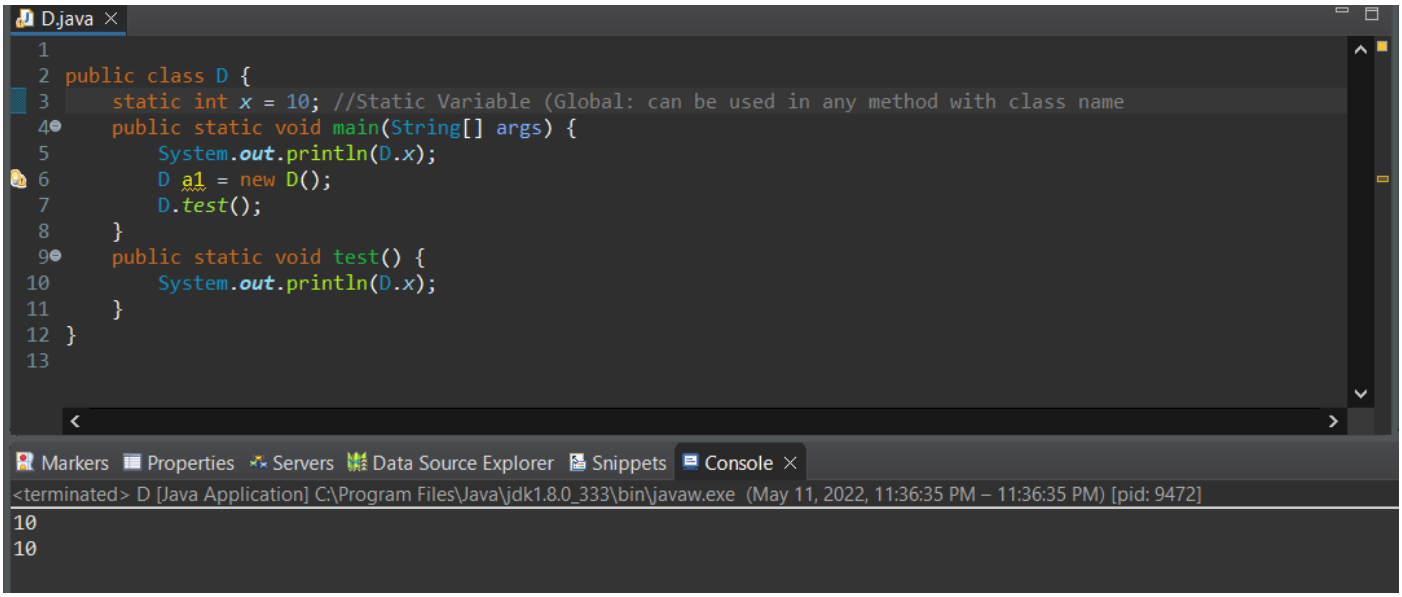
```
1
2 public class D {
3     public static void main(String[] args) {
4         int x ; //did not initialized (no value given)
5         System.out.println(x);
6     }
7 }
8
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> D [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 11, 2022, 4:31:47 PM – 4:31:47 PM) [pid: 10960]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The local variable x may not have been initialized

        at D.main(D.java:5)
```

2. Static Variable: These variables are global and can be used in any method with class name.

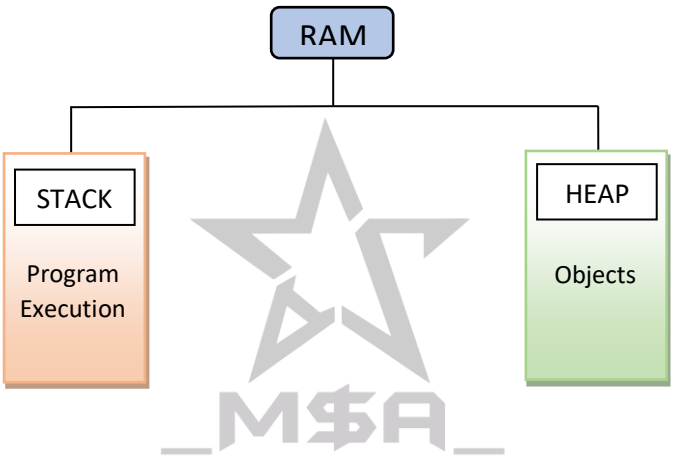


27/04/2022 (Wednesday)

STACK AND HEAP MEMORY

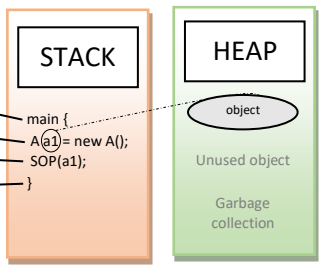
STACK AND HEAP MEMORY:

- Program execution flow is maintained in stack & removed in last-in-first
- All the objects created in java are stored in heap, and after the execution of program the reference to the object is gone, and object without reference in JAVA is called as "unused object" and is eligible for garbage collection.
- Non static & static variables are stored in a dedicated space in stack memory called "META SPACE", It stores the original copy of static and non static variable, It doesn't store the local variables. Non static variables from meta space gets loaded into object where as the Static variable remains there.



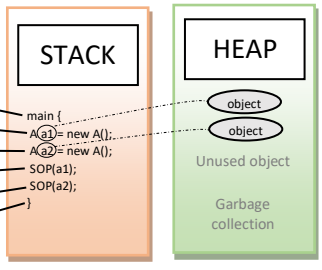
Example 1:

```
public class A {
        public static void main(String[] args) {
                A a1 = new A();
                System.out.println(a1);
        }
}
```



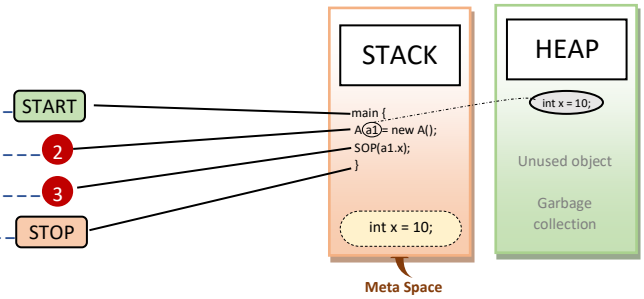
Example 2:

```
public class A {
        public static void main(String[] args) {
                A a1 = new A();
                A a2 = new A();
                System.out.println(a1);
                System.out.println(a2);
        }
}
```



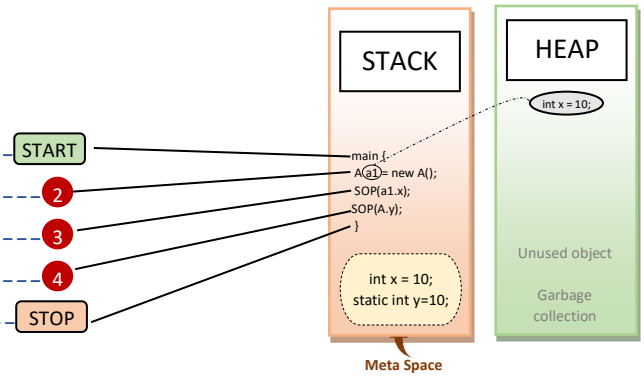
Example 3: *(with non static variable)*

```
public class A {
    int x = 10;     //non static variable
        public static void main(String[] args) {
                A a1 = new A();
                System.out.println(a1.x);
        }
}
```



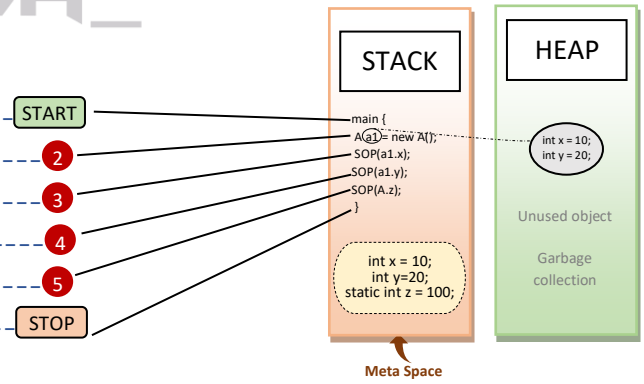
Example 4: *(with non static & static variable)*

```
public class A {
    int x = 10; //non static variable
    static int y = 20; //static variable
        public static void main(String[] args) {
                A a1 = new A();
                System.out.println(a1.x);
                System.out.println(A.y);
        }
}
```



Example 5: *(with non static & static variable)*

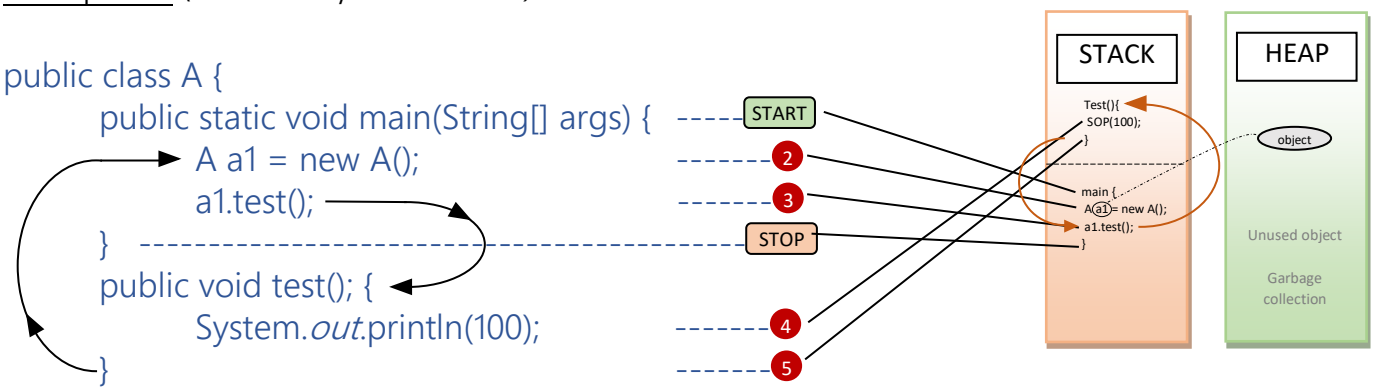
```
public class A {
    int x = 10; //non static variable
    int y = 20; //non static variable
    static int z = 100; //static variable
        public static void main(String[] args) {
                A a1 = new A();
                System.out.println(a1.x);
                System.out.println(a1.y);
                System.out.println(A.z);
        }
}
```



28/04/2022 (Thursday)

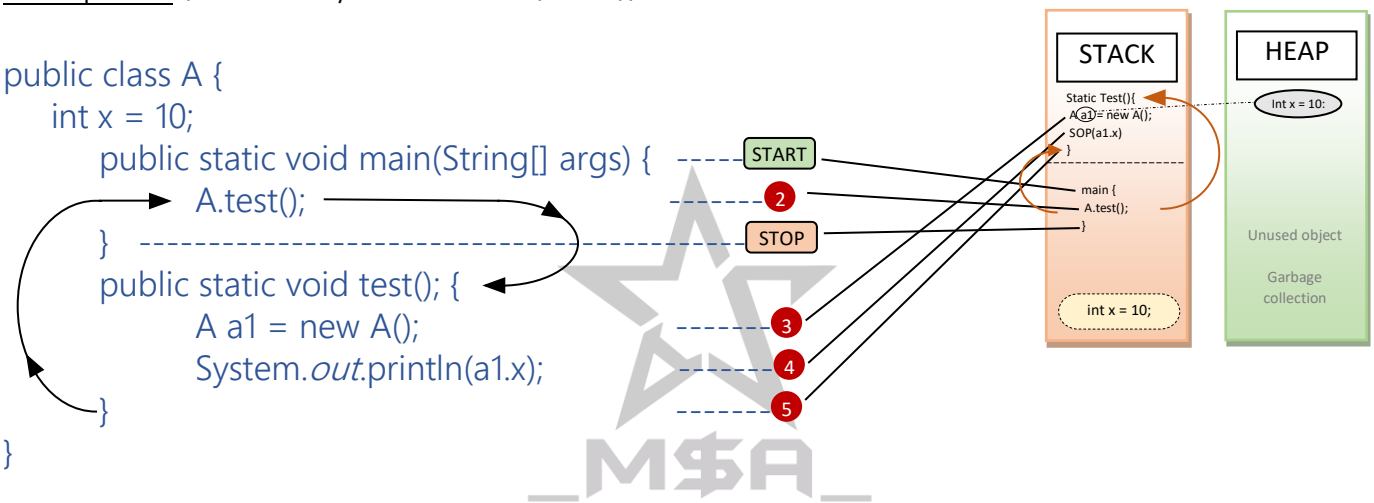
Example 6: *(with multiple methods)*

```
public class A {
        public static void main(String[] args) {
                A a1 = new A();
                a1.test();
        }
        public void test(); {
                System.out.println(100);
        }
}
```



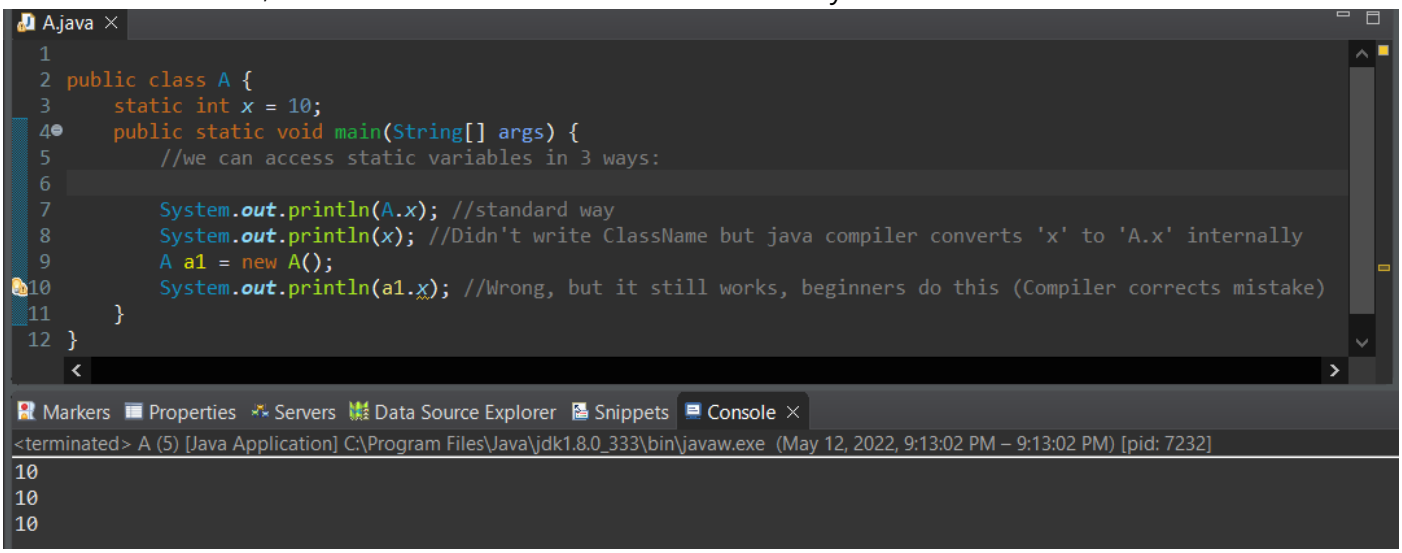
Example 7: *(with multiple methods (static))*

```
public class A {
    int x = 10;
        public static void main(String[] args) {
                A.test();
        }
        public static void test(); {
                A a1 = new A();
                System.out.println(a1.x);
        }
}
```

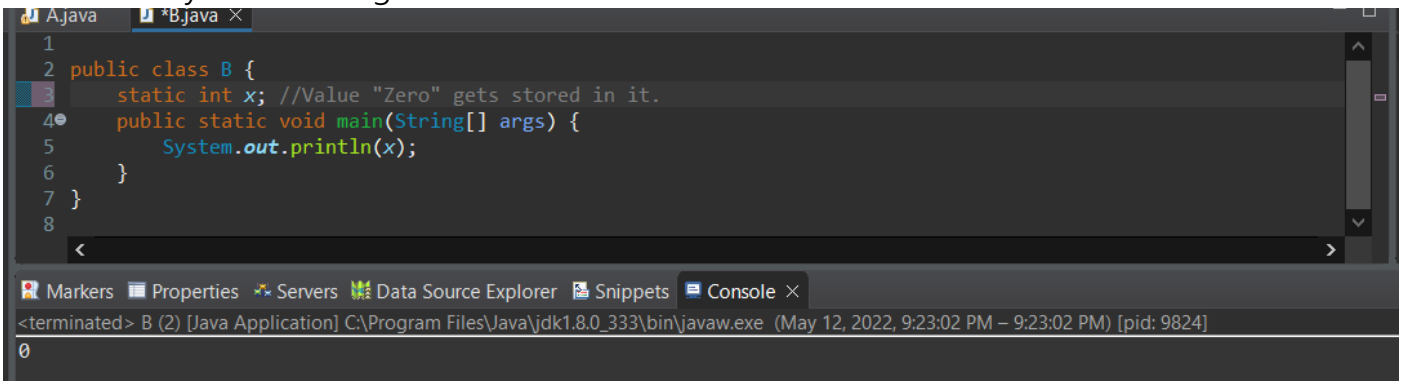


Types of variables in Java (Contd…)

1. Static Variable (Contd..): These variables are global and can be used in any method with ClassName, we can access static variables in 3 ways.



It is not mandatory to initialize static variable. if we don't initialize then depending on the 'data type' default value gets stored in it. In the below example because the data type is "int", automatically value zero gets stored in it.



29/04/2022 (Friday)

DATA TYPES IN JAVA

Data Type	Memory Size	Default Value	Value Range	Value Type
Byte	1 byte	0	-128 .. 127	Integer Value
Short	2 bytes	0	-32,768 .. 32,767	
Int	4 bytes	0	-2,147,483,648 .. 2,147,483,647	
Long	8 bytes	0	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807	
Float	4 bytes	0.0	$3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$	Decimal Value
Double	8 bytes	0.0		
Char	2 bytes	Empty Space		Character
Boolean	NA	false		True/False
String (Class)	NA	null		
Var (Ver. 10 of Java)	Dynamic			

Integer default value Example:

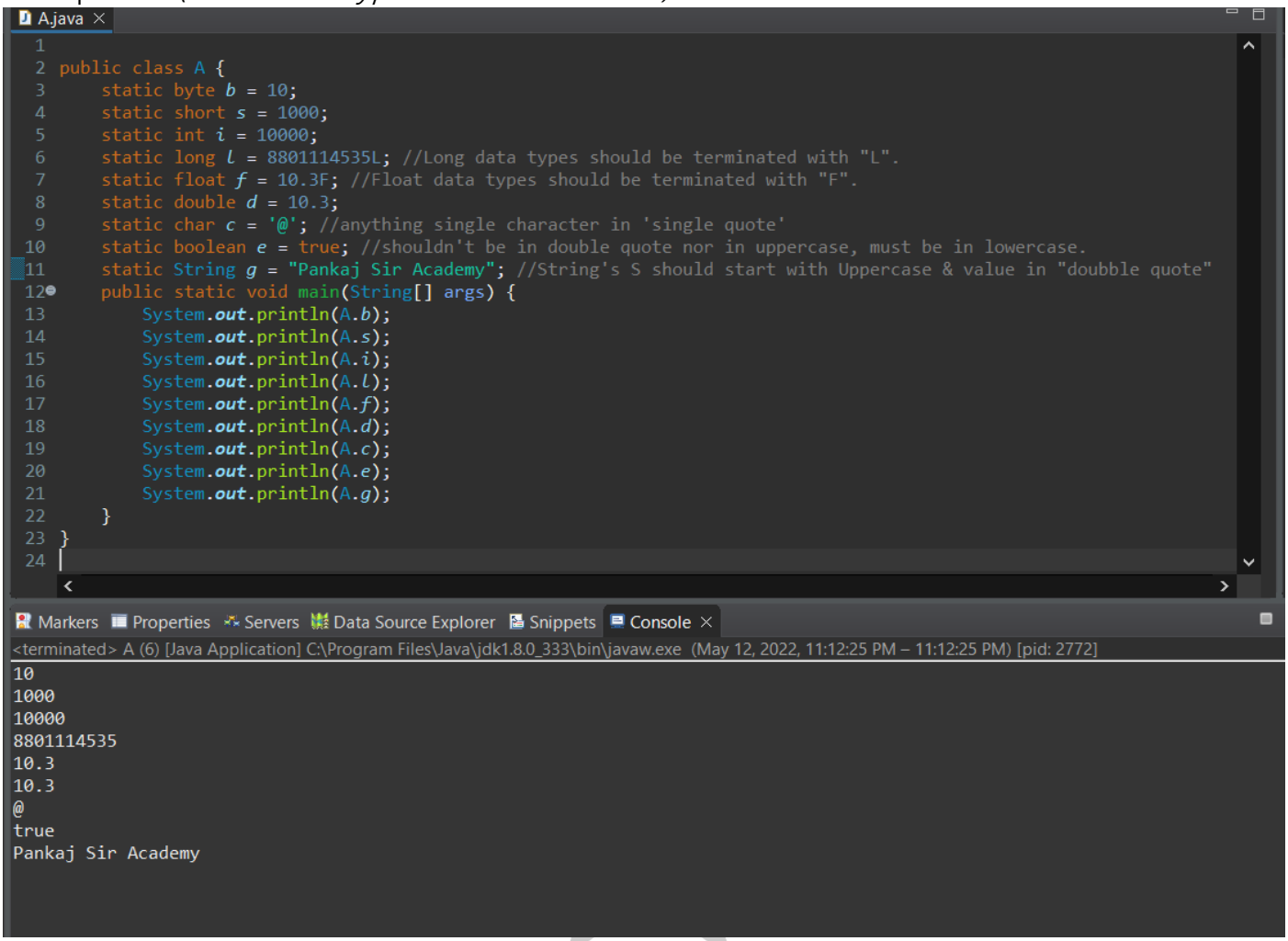
A.java

```
1
2  public class A {
3      static byte b; //0
4      static short s; //0
5      static int i; //0
6      static long l; //0
7      public static void main(String[] args) {
8          System.out.println(A.b);
9          System.out.println(A.s);
10         System.out.println(A.i);
11         System.out.println(A.l);
12     }
13 }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (6) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 12, 2022, 10:23:04 PM – 10:23:05 PM) [pid: 2660]
0
0
0
0
```

Example #2: *(all the data types with some values)*



A.java

```
1
2  public class A {
3      static byte b = 10;
4      static short s = 1000;
5      static int i = 10000;
6      static long l = 8801114535L; //Long data types should be terminated with "L".
7      static float f = 10.3F; //Float data types should be terminated with "F".
8      static double d = 10.3;
9      static char c = '@'; //anything single character in 'single quote'
10     static boolean e = true; //shouldn't be in double quote nor in uppercase, must be in lowercase.
11     static String g = "Pankaj Sir Academy"; //String's S should start with Uppercase & value in "doubble quote"
12     public static void main(String[] args) {
13         System.out.println(A.b);
14         System.out.println(A.s);
15         System.out.println(A.i);
16         System.out.println(A.l);
17         System.out.println(A.f);
18         System.out.println(A.d);
19         System.out.println(A.c);
20         System.out.println(A.e);
21         System.out.println(A.g);
22     }
23 }
24
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

<terminated> A (6) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 12, 2022, 11:12:25 PM – 11:12:25 PM) [pid: 2772]

```
10
1000
10000
8801114535
10.3
10.3
@
true
Pankaj Sir Academy
```

CONVENTIONS IN JAVA

- All the KEYWORDS in java are in lowercase.
  Ex: new, class, static, int, short

- ClassName should start with Uppercase
  Ex: Bank.
  public class Bank {
  If there are two words in ClassName it should be in CamelCasing
  Ex: BankAccount, (it improves readability)
  public class BankAccount {

- Variable name should be in lowercase or start with lowercase, if it has two words, it should be in camelCasing. 1st word start with lowercase & the rest start with uppercase.
  Ex: int id;
  int employeeId;
  int yourIdCard;

- Variable name cannot start with number in Java.

  Ex: int 2id; *//Error*
  int id2: *//Correct*

- Can not create variable name with space

  Ex: Your Card; *//Error*

- Only two special characters are allowed in Java, UnderScore "_" and Dollar "$", we can use it anywhere, no other special characters are allowed.

  Ex: int $id;
  int i$d;
  int id$;
  int your_card;
  int _your_card;

- Methods in Java can be identified with parentheses "()" beside variable and in casing it is same as variables.

  Ex: employeeId()
  yourCardId()

04/05/2022 (Wednesday)

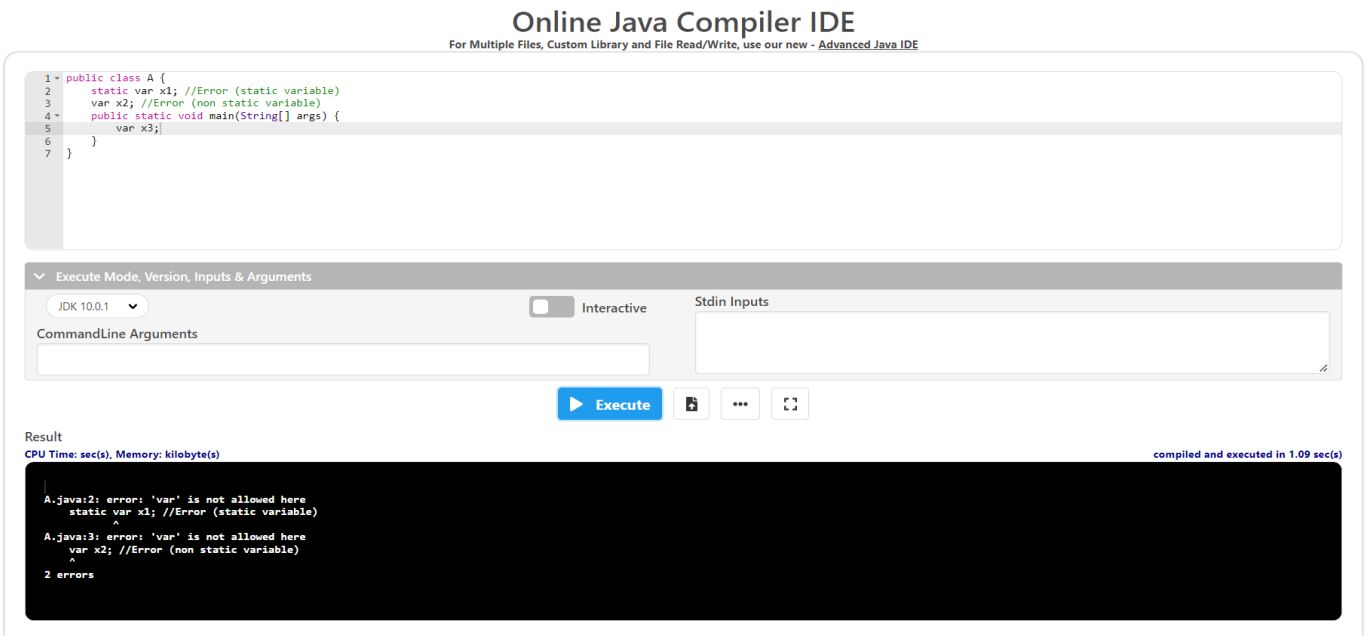
VAR TYPES IN JAVA

Var type was introduced in version 10 of Java, it will not work with Java below V10. A variable with the type 'var' can store any kind of value in it. It cannot be static or non static variable, it can only be local variable & created inside method.

Example:

Limitations of var type:

A variable with type var can not be a method argument and static or non static variable. It can only be local variable as shown in the below example:

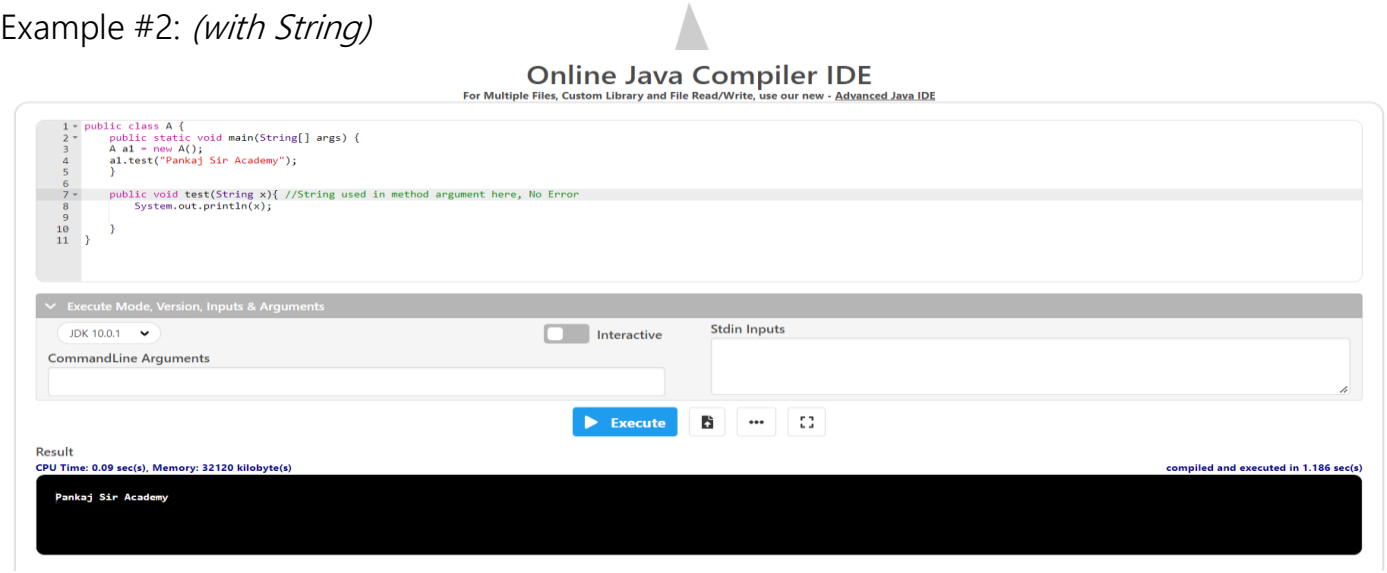


The memory size of var type is dynamic, means depending upon the value stored the memory increases and decreases.

A var type can not be method argument, see the below examples:

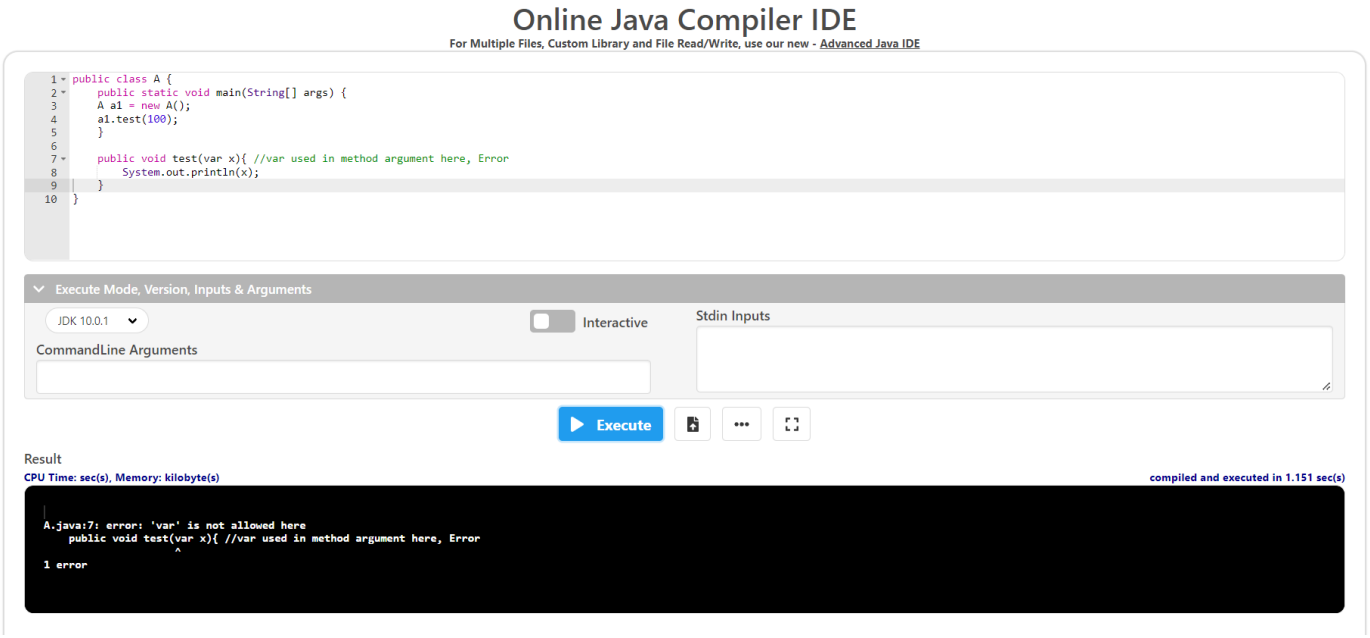
Example #1: *(with data type int)*

Example #2: *(with String)*



Example #2: *(with int and String)*

Example #3: *(using var in method argument)*



Var is not a keyword in Java & hence we can use it as variable name as well, Ex:

Types of variables in Java (Contd…)

4. Reference Variable: In a reference variable we can store either object's address or null. The data type of variable name is ClassName.

Data type	Variable Name	Assgn. Operator	Value	Terminator
int	x	=	10	;
char	c	=	'a'	;

While creating reference variable

ClassName	Variable Name	Assgn. Operator	Object's address / null	Terminator
A	x	=	new A ()	;
A	x	=	null	;

Types of reference variables:

1) Local reference variable
2) Static reference variable

1. Local Variable: These variables are created inside method & should be used only within the created method. If used outside method the program will throw an error, see the below example:

2. Static reference variable: These variables are created outside method but inside class using static keyword. This variable has global access & can be used anywhere in the program.

The default value for a static reference variable is 'null' as shown in below example:

06 /05/2022 (Friday)

METHODS IN JAVA

1) void
2) return value
3) return

1. void method: if a method is void, then it can not return any value, hence the below program throws an error:

```
A.java
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         a1.test();
6     }
7
8     public void test() {
9         return 100; //Error
10    }
11 }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 1:06:19 PM – 1:06:19 PM) [pid: 2896]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Void methods cannot return a value

        at A.test(A.java:9)
        at A.main(A.java:5)
```

Example #2: *without void method (if return value is needed)*:
If return value is needed, we should use data type of the return value in method argument.

```
A.java
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         int x = a1.test(); //before 'x' we're writing return value's data type, & then calling the method.
6         System.out.println(x); //
7     }
8     public int test() { //here there's no 'void' but the 'int' because the returning value is integer.
9         return 100;
10    }
11 }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 2:21:22 PM – 2:21:23 PM) [pid: 3796]
100
```

Example #3:

A.java

```
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         double x = a1.test(); //before 'x' we're writing return value's data type, & then calling the method.
6         System.out.println(x); //
7     }
8     public double test() { //here there's no 'void' but 'double' because the returning value is integer.
9         return 10.3;
10    }
11 }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 2:25:24 PM – 2:25:25 PM) [pid: 18752]
10.3
```

Example #4:

*A.java

```
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         String x = a1.test(); //before 'x' we're writing String, & then calling the method.
6         System.out.println(x); //
7     }
8     public String test() { //here there's no 'void' but 'String' because the returning value is text.
9         return "Pankaj Sir Academy";
10    }
11 }
```

Markers Properties Servers Data Source Explorer Snippets Console

```
<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 2:27:31 PM – 2:27:31 PM) [pid: 8392]
Pankaj Sir Academy
```

Example #5:

A.java

```
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         boolean x = a1.test(); //before 'x' we're writing boolean, & then calling the method.
6         System.out.println(x); //
7     }
8     public boolean test() { //here there's no 'void' but 'boolean' because the returning value is true.
9         return true;
10    }
11 }
```

Markers Properties Servers Data Source Explorer Snippets Console

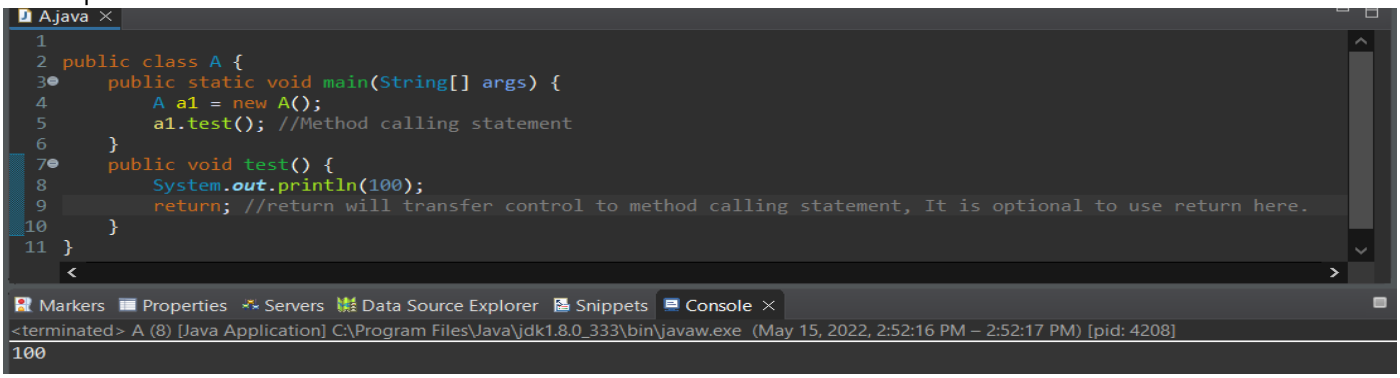
```
<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 2:48:07 PM – 2:48:08 PM) [pid: 16616]
true
```

Q. What's the difference between 'return' and 'return value' .? ***

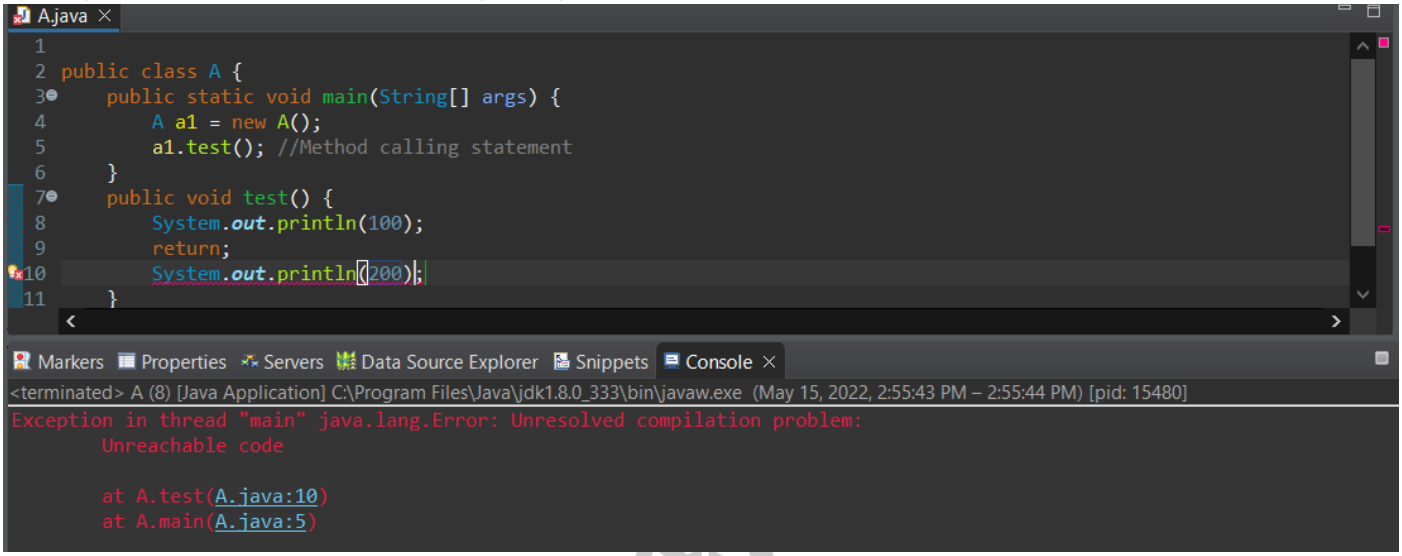
A.

return		return value	
1.	We can use return keyword inside void methods only.	1.	We can use return value statement inside not a void method.
2.	It's optional to use inside method.	2.	If a method is not a void then, writing return value is mandatory.
3.	return keyword will transfer control to method calling statement.	3.	It will transfer control and value both to the method calling statement.

Example #1:

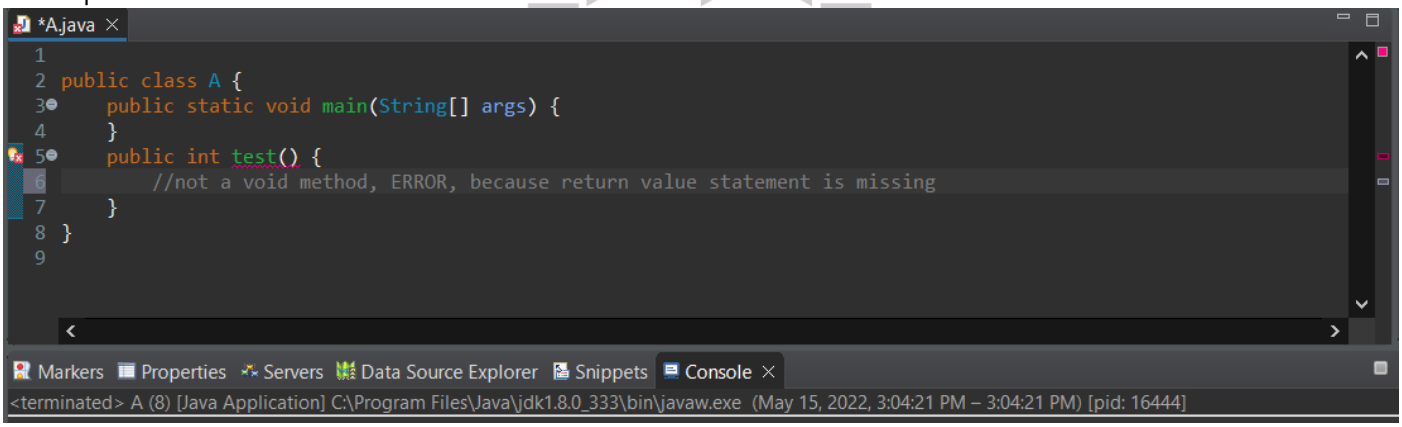


Example #2: *We cannot write anything after 'return', (unreachable code error):*



If a method is not a void, then writing return value is mandatory.

Example #3:



Example: *Multiple method arguments: the number of values, value's data type and order should match in method argument.*

A.java

```java

public class A {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(10, "mike", true, 'a', 10.3);
    }
    public void test(int i, String s, boolean b, char c, double d) {
        System.out.println(i);
        System.out.println(s);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Console

<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 3:15:15 PM – 3:15:15 PM) [pid: 1460]

```
10
mike
true
a
10.3
```

Example #2: *multiple single data type values & single variable in method argument:*

*A.java

```java

public class A {
    public static void main(String[] args) {
        A a1 = new A();
        a1.test(10, 20, 30, 40, 50);
    }

    public void test(int... i) { //Written 3 dots after int.
        System.out.println(i[0]);
        System.out.println(i[1]);
        System.out.println(i[2]);
        System.out.println(i[3]);
        System.out.println(i[4]);
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Console

<terminated> A (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 15, 2022, 3:24:09 PM – 3:24:10 PM) [pid: 12576]

```
10
20
30
40
50
```

09/05/2022 (Monday)

CONSTRUCTORS IN JAVA

In Java, a constructor is a block of codes similar to the method. It is called when an object/instance of the class is created.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, constructor gets called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

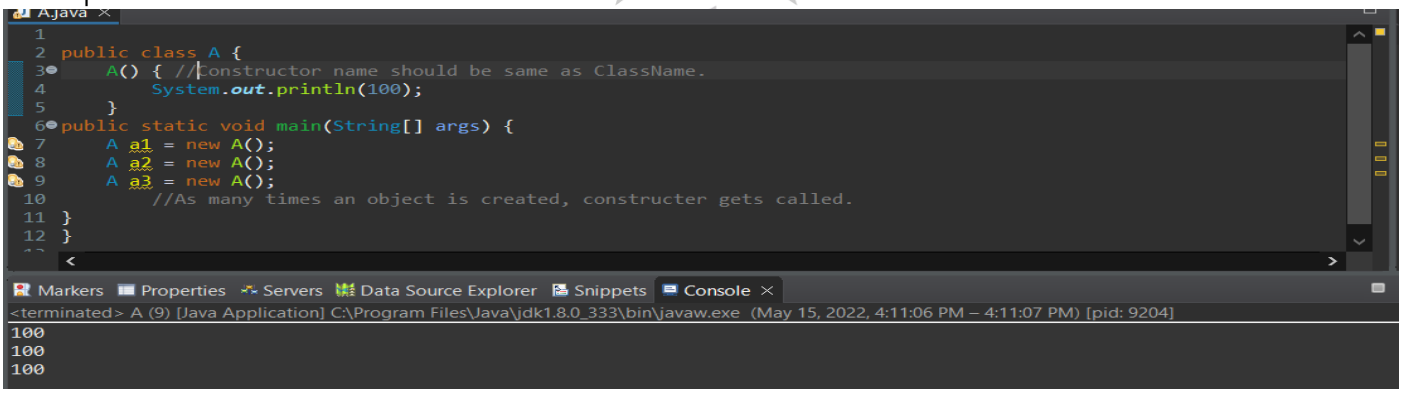
Rules of JAVA Constructors:

- Constructors should have same name as that of Class.
- Constructors are permanently void.
- A Java constructor cannot be abstract, static, final, & cannot return any value.

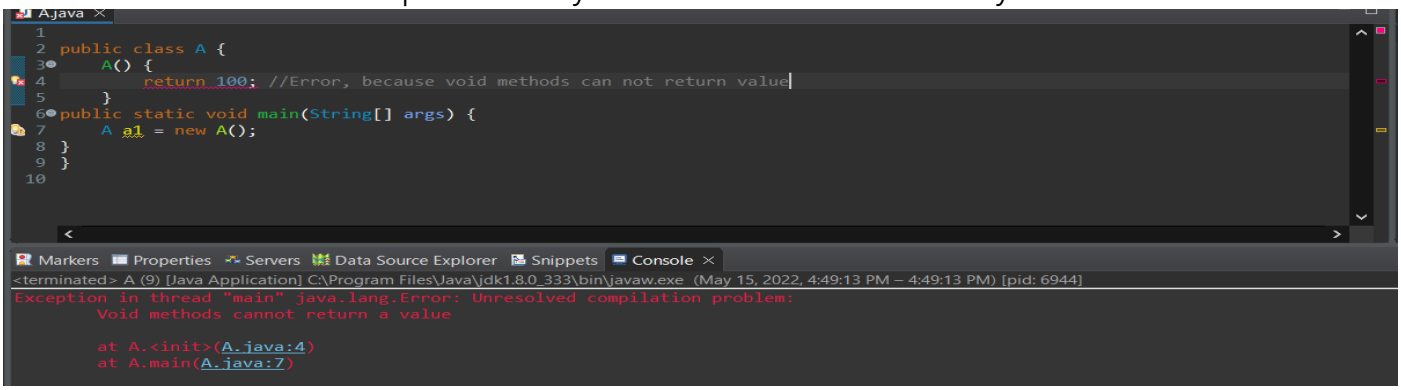
Q. What's the difference between "constructor" and "method" ?

A. Constructors are special methods, constructors are permanently void but methods can be void & can not be void. Constructors are called only when object is created, methods are called either with ClassName or after creating an object, ultimately both are methods only.

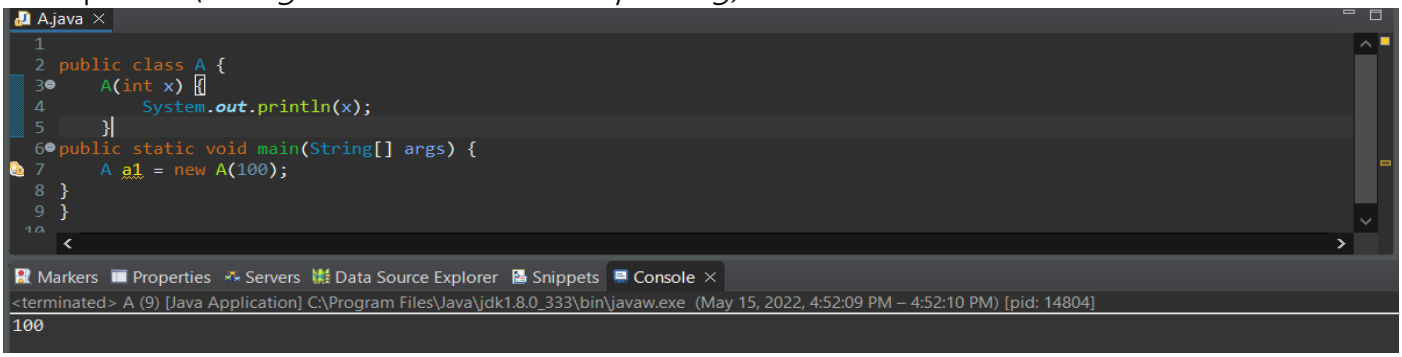
Example #1:



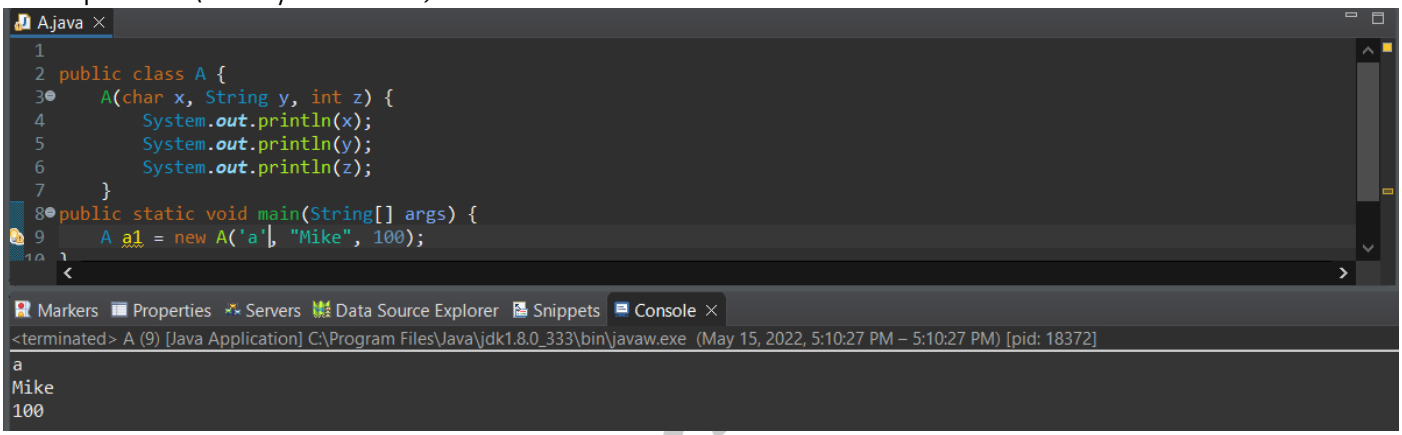
- Constructors are permanently void & can never return any value



Example #2: *(Giving some value to x and printing)*



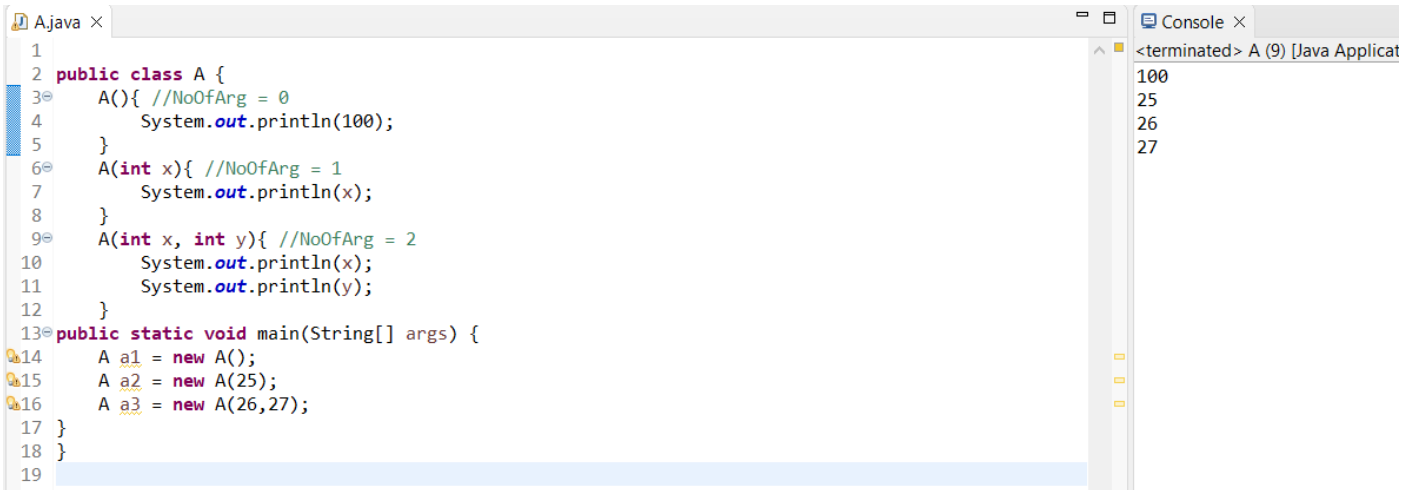
Example #3: *(multiple values)*



CONSTRUCTOR OVERLOADING:

Here we create more than one constructor in same class provided they are different no. of arguments & different type of arguments.

Example #1: *(different no. of arguments)*



Example #2: *(different type of arguments)*

A.java

```
1
2  public class A {
3      A(int x){ //NoOfArg = 1, type = int
4          System.out.println(x);
5      }
6      A(char x){ //NoOfArg = 1, type = char
7          System.out.println(x);
8      }
9      A(boolean x){ //NoOfArg = 1, type = boolean
10         System.out.println(x);
11     }
12     A(double x, String y){ //NoOfArg = 2, type=double & String
13         System.out.println(x);
14         System.out.println(y);
15     }
16 public static void main(String[] args) {
17     A a1 = new A(100);
18     A a2 = new A('a');
19     A a3 = new A(true);
20     A a4 = new A(25.6, "Pankaj Sir Academy");
21 }
22 }
```

Console

```
<terminated> A (9) [Java Applicat
100
a
true
25.6
Pankaj Sir Academy
```

Never use void or data type (int, char float etc..) while creating a constructor, If you do that then it is a method and not a constructor.

Example #1:

A.java

```
1
2  public class A {
3      void A() { //Method, non static
4          System.out.println(100);
5      }
6      public static void main(String[] args) {
7          A a1 = new A();
8
9      }
10 }
11 // Output will not call "void A()" and will print no output.
```

Console

```
<terminated> A (9) [Java Application] C:\Program Files
```

Example #2:

A.java

```
1
2  public class A {
3      void A() { //Method, non static
4          System.out.println(100);
5      }
6      public static void main(String[] args) {
7          A a1 = new A();
8          a1.A();
9      }
10 }
11 // 100
```

Console

```
<terminated> A (9
100
```

Example #3:

A.java Run A (9)

```
1
2  public class A {
3      int A() { //Method, non static
4          return 10;
5      }
6      public static void main(String[] args) {
7          A a1 = new A();
8          int val = a1.A();
9          System.out.println(val);
10     }
11 }
12
```

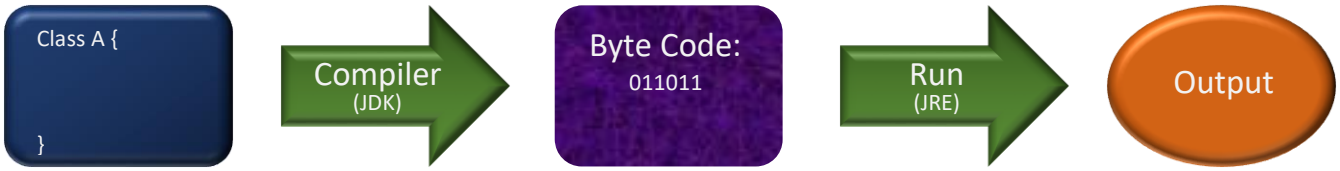
Console

```
<terminated> A (9
10
```

10/05/2022 (Tuesday)

Default Constructors:

Every character has a Unicode value in numerical representation in Java.
For example:
a = 99
b = 98
c = 99 etc...



File: A.java File: A.class

".java" file is compiled into ".class" file and ".class" file consist of byte code which we further run ".class" file to get the output.
If you are a developer, we install JDK which gives compiler and runtime environment.
If you are a customer, we install only JRE because we run only ".class" file. Whenever we create an object, it has to mandatorily call constructor, if a constructor is not created explicitly by developer, then during compilation automatically empty body constructor gets added in dot ".class" file. This is applicable only for no argument constructor and these constructors are called as default constructors.

A.java

*A.java

```
1  package p1;
2
3  public class A {
4
5      //no constructor is created
6
7      public static void main(String[] args) {
8          A a1 = new A();
9      }
10 }
11
```

A.class

*A.java

```
1  package p1;
2
3  public class A {
4      A(){
5      //default constructor created automatically
6      }
7      public static void main(String[] args) {
8          A a1 = new A();
9      }
10 }
11
```

When an object with argument is created and if matching constructor is missing in ".java" file then it shows an error as shown in the below example:

A.java

```
1  package p1;
2
3  public class A {
4      public static void main(String[] args) {
5          A a1 = new A(100); //Error
6      }
7  }
8
```

Console

```
<terminated> A (10) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 18, 2022, 1
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The constructor A(int) is undefined

        at p1.A.main(A.java:5)
```

When an object with argument is created & an object without argument is created in the same program then the concept of default constructor is not applicable, hence the below programs throws an error:

*A.java

```
1  package p1;
2  public class A {
3      A (int x) {
4
5      } //Only one constructor is created
6      public static void main(String[] args) {
7          A a1 = new A(100);
8          A a2 = new A(); //Error
9      }
10 }
11
```

Console

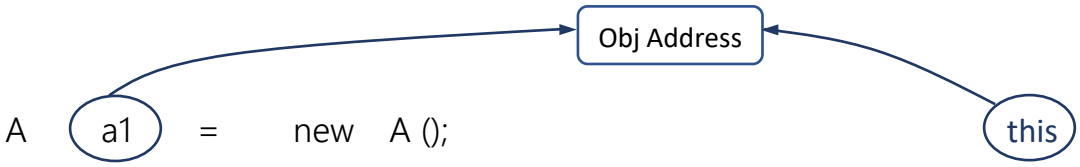
```
<terminated> A (10) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 18, 2022, 10:23:43
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The constructor A() is undefined

        at p1.A.main(A.java:8)
```

11/05/2022 (Wednesday)

"this" (keyword):

It's a special reference variable that gets created automatically.



Example:

*A.java

```
1
2  public class A {
3      public static void main(String[] args) {
4          A a1 = new A();
5          System.out.println(a1);
6          a1.test();
7      }
8      public void test() {
9          System.out.println(this);
10     } //prints same address twice
11
12 }
13
```

Console

```
<terminated> A (1
A@15db9742
A@15db9742
```

Using "this" keyword we can access non-static members of class.

Example #2:

*A.java

```
1
2  public class A {
3      int x = 10;
4      public static void main(String[] args) {
5          A a1 = new A();
6          System.out.println(a1.x);
7          a1.test();
8      }
9      public void test() {
10         System.out.println(this.x);
11     } //prints same address twice
12 }
```

Console

```
<terminated> A (11) [
10
10
```

Example #3:

*A.java

```
1
2  public class A {
3      public static void main(String[] args) {
4          A a1 = new A();
5          a1.test1();
6      }
7      public void test1() {
8          this.test2();
9      }
10     public void test2() {
11         System.out.println(500);
12     }
13 }
```

Console

```
<terminated> A (11) [Java Application]
500
```

We cannot use "this" keyword inside static methods.

Example: #4

*A.java

```
1
2 public class A {
3     public static void main(String[] args) { //static method
4         A a1 = new A();
5         System.out.println(this); //Error
6     }
7 }
```

Console

```
<terminated> A (11) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 19, 2022
Exception in thread "main" java.lang.Error: Unresolved compilation problem
        Cannot use this in a static context

        at A.main(A.java:5)
```

Example #5:

A.java

```
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A();
5         a1.test();
6     }
7     public static void test() { // static method
8         System.out.println(this); //Error because static method
9     }
10 }
```

Console

```
<terminated> A (11) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 19, 202
Exception in thread "main" java.lang.Error: Unresolved compilation problem
        Cannot use this in a static context

        at A.test(A.java:8)
        at A.main(A.java:5)
```

If we do not use "this" keyword inside non static method to access non static members then during compile time "this" keyword gets appended automatically.

Example #6:

A.java

```
1
2 public class A {
3     int x = 10;
4     public static void main(String[] args) {
5         A a1 = new A();
6         a1.test();
7     }
8     public void test() {
9         System.out.println(this.x);
10        System.out.println(x);
11        //we can't use (a1.x) because it's local variable, but can access through (this.x)
12        //if we don't write "this" keyword, it'll be appended automatically
13    }
14 }
```

Console

```
<terminated> A (11) [Java Applica
10
10
```

If there are multiple objects created then "This" keyword points to current object running in the program.

Example #7:

*A.java

```
1
2 public class A {
3     public static void main(String[] args) {
4         A a1 = new A(); //
5         a1.test(); //running object a1
6         A a2 = new A();
7         a2.test(); //running object a2
8     }
9     public void test() { //we called "this" twice, so it printed what object was running at that moment.
10        System.out.println(this);
11    }
12 }
```

Console

```
<terminated> A (
A@15db9742
A@6d06d69c
```

Using “this” keyword we can access static members, but we should not do that.

Example#8:

A.java

```
1
2  public class A {
3      static int x = 10;
4      public static void main(String[] args) {
5          A a1 = new A();
6          a1.test();
7      }
8      public void test() { //non static method
9          System.out.println(this.x); //Warning.!! It should be accessed in static way.
10         //but still during compilation "this.x" is automatically replaced with "A.x"
11     }
12 }
```

Console

```
<terminated> A (
10
```

12/05/2022 (Thursday)

"this" (keyword): (Contd...)

There are two usages of "this" keyword:

- *this.memberName;* (The member here is a non static member).
- *this();* (It is used to call constructor but this call should happen from another constructor.

Example:

A.java

```
1  public class A {
2      A(){ //Constructor #1
3          System.out.println(100);
4      }
5      A(int x){ //Contsructor #2
6          this(); //Calling Constructor #1
7      }
8      public static void main(String[] args) {
9          A a1 = new A(100);
10     }
11 }
12
```

Console

```
<terminated> A (
100
```

This is also called as constructor chaining.

Q. What is constructor chaining?

A. When we call one constructor from another constructor is called as constructor chaining.

While calling constructor from another constructor "this()" keyword should always be First statement.

Example #1:

A.java

```
1  public class A {
2      A(){ //Constructor #1
3          System.out.println(100);
4      }
5      A(int x){ //Contsructor #2
6          this(); //First Statement, Correct
7          System.out.println(100); //Second Statement
8          this(); //Third Statement, Error
9      }
10     public static void main(String[] args) {
11         A a1 = new A(100);
12     }
13 }
```

Console

```
<terminated> A (12) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2022,
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Constructor call must be the first statement in a constructor

        at A.<init>(A.java:8)
        at A.main(A.java:11)
```

Example #2:

A.java

```
1  public class A {
2      A(){
3          this(100); //We're calling Line #5
4      }
5      A(int x){
6          System.out.println(100);
7      }
8      public static void main(String[] args) {
9          A a1 = new A(); //We're calling line #2
10     }
11 }
```

Console

```
<terminated> A (12)
100
```

Example#3:

A.java

```
1  public class A {
2      int x ; //non static variable
3      public static void main(String[] args) {
4          A a1 = new A();
5          a1.test();
6          System.out.println(a1.x);
7      }
8      public void test() { //non static method
9          int x = 10; //Local variable
10         this.x = x; //"this.x" is calling non static variable (line #2), and "x" value copied from local variable.
11     }
12 }
```

Console

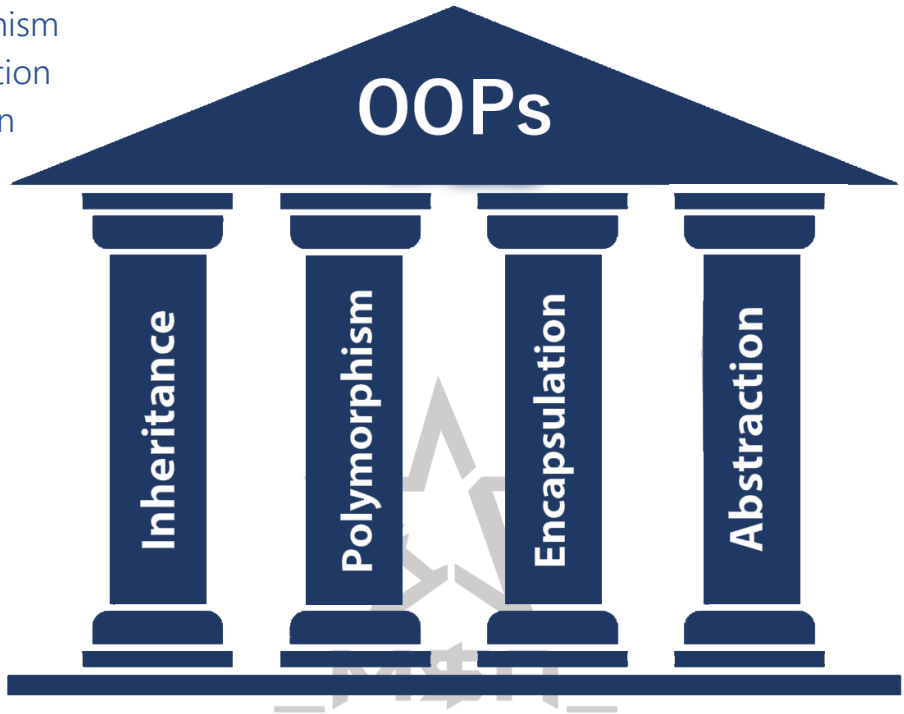
```
<terminated>
10
```

13/05/2022 (Friday)

OBJECT ORIENTED PROGRAMMING (OOPs) CONCEPT IN JAVA

The following are the Four Pillers of OOPs.

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction



Inheritance: Here we inherit the members of parent class (super class) to child class with an intension of re-using it. Example #1:

B.java

```
1 public class B extends A { //ChildClass
2     public static void main(String[] args) {
3         B b1 = new B(); //In this object the value of "x" from Class A is stored.
4         System.out.println(b1.x);
5     }
6 }
```

A.java

```
1 public class A { //ParentClass (SuperClass)
2     int x = 10;
3 }
4
```

Console

```
<terminated> B (3) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 23, 2022, 10:25:10 AM – 10:25:
10
```

Example #2:

B.java

```
1 public class B extends A { //ChildClass
2     public static void main(String[] args) {
3         B b1 = new B(); //In this object the value of "x" from Class A is stored.
4         System.out.println(b1.x);
5         b1.test(); //Method called from Class A.
6     }
7 }
```

A.java

```
1 public class A { //ParentClass (SuperClass)
2     int x = 10;
3
4     public void test() {
5     System.out.println(100);
6     }
7 }
```

Console

```
<terminated> B (3) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 23, 2022, 10:32:35 AM – 10:32:
10
100
```

Example #3: *(Inheritance between more than 2 class)*

C.java

```
1  public class C extends B { //ChildClass2
2      //test1, test2, test3
3      public void test3() {
4          System.out.println(300);
5      }
6      public static void main(String[] args) {
7          C c1 = new C();
8          c1.test1();
9          c1.test2();
10         c1.test3();
11     }
12 }
```

B.java

```
1  public class B extends A { //ChildClass
2      //test1, test2
3      public void test2() {
4          System.out.println(200);
5      }
6  }
```

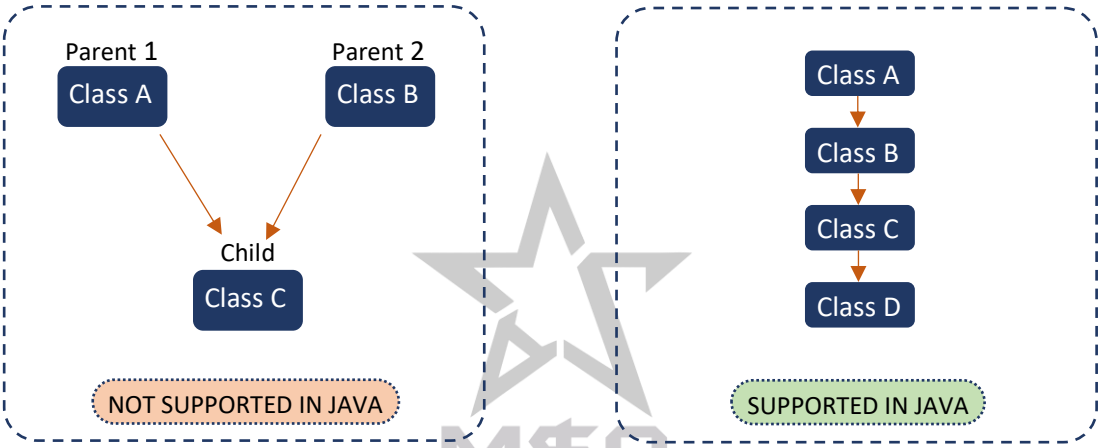
Console

```
<terminated> C (1) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2022, 10:43:43 AM – 10:43:
100
200
300
```

A.java

```
1  public class A { //ParentClass (SuperClass)
2      //test1
3      public void test1() {
4      System.out.println(100);
5      }
6  }
```

Note: At Class-level Java doesn't support multiple inheritances, but at Interfaces-level it supports multiple inheritances.



Example:

C.java

```
1  public class C extends B,A { //Error
2
3  }
```

B.java

```
1  public class B extends A {
2
3  }
```

A.java

```
1  public class A {
2
3  }
```

Assignment

Watch YouTube Playlist by Pankaj Sir Academy Chanel entitled

"Unary Operator in JAVA"

& Prepare notes.

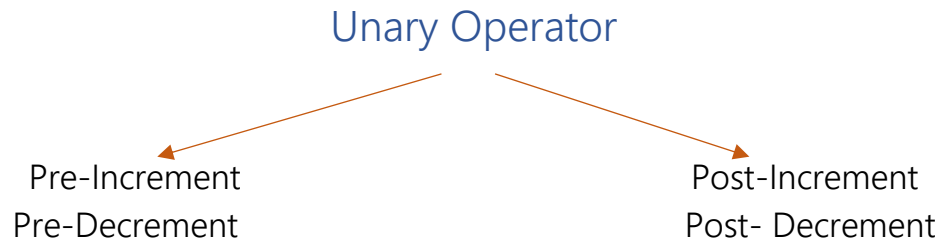
Click below to watch on YouTube:

YouTube Click Here

14/05/2022 (Saturday) : Assignment – 1

UNARY OPERATOR IN JAVA

In Java, the unary operator is an operator that can be used only with an operand. It is used to represent the positive or negative value, increment/decrement the value by 1.



- *Post-Increment Examples*

Example #1:

*A.java

```
1  public class A {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = i++ + i++; //post-increment (beacause after variable)
5              //"++" means increase the value by 1 only when we see the same variable next time.
6          System.out.println(i); //here again we see "i" variable,
7          System.out.println(j);
8              //first i++=10, second i++=11 and third i=12
9      }
10 }
```

Console

```
<terminated> A (13
12
21
```


Example #2:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = i++ + i++ + i++;
5
6          System.out.println(i); //13
7          System.out.println(j); //10+11+12=33
8          ////first i++=10, second i++=11, third i++=12 and fourth i= 13.
9      }
10 }
```

Console

```
<terminated> B (4
13
33
```

Example #3:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = i++ + i++ + i + i++;
5          //       10    11    12  12
6          //here we didn't apply ++ in third i so it means no increment when we see next time
7          //so value remains same as previous i++
8          System.out.println(i); //13
9          System.out.println(j); //10+11+12+12=45
10     }
11 }
```

Console

```
<terminated> B
13
45
```

Example #4:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = i++ + i++ + i + i++ +i + i++;
5          //      10    11    12  12   13  13
6          System.out.println(i); //14
7          System.out.println(j); //10+11+12+12+13+13=71
8      }
9  }
```

Console

```
<terminated> B
14
71
```

- *Pre-Increment Examples*

Example #1:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = ++i + ++i;
5          //      11     12
6          //Before i there's "++", It means increment in the same step, no increment in post variable.
7          System.out.println(i); //12
8          System.out.println(j); //11+12=23
9      }
10 }
```

Console

```
<terminated> B
12
23
```

Example #2:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = ++i + i + ++i;
5          //      11    11   12
6          System.out.println(i); //12
7          System.out.println(j); //11+11+12=34
8      }
9  }
```

Console

```
<terminated> B
12
34
```

- *Pre and post increments (both mixed) Examples*

Example #1:

A.java B.java

```
1  public class B {
2      public static void main(String[] args) {
3          int i = 10;
4          int j = ++i + i++;
5          //      11     11
6          System.out.println(i); //12
7          System.out.println(j); //11+11=22
8      }
9  }
```

Console

```
<terminated> B
12
22
```

Example #2:

```java
public class B {
    public static void main(String[] args) {
        int i = 10;
        int j = ++i + i + i++ + ++i + i++;
        //      11    11  11     13    13
        System.out.println(i); //14
        System.out.println(j); //11+11+11+13+13=59
    }
}
```

```
<terminated> B
14
59
```

After understanding Increment concept, it's easier to understand Decrement Concept, in increment we increased the value by 1 but in decrement we will decrease the value by 1.

- *Post-Decrement Examples*

Example #1:

```java
public class B {
    public static void main(String[] args) {
        int i = 10;
        int j = i-- + i--;
        //      10    9
        System.out.println(i); //8
        System.out.println(j); //19
    }
}
```

```
<terminated> B
8
19
```

Example #2:

```java
public class B {
    public static void main(String[] args) {
        int i = 10;
        int j = --i + --i;
        //      9      8
        System.out.println(i); //8
        System.out.println(j); //9+8=17
    }
}
```

```
<terminated> B
8
17
```

All mixed example:

```java
public class B {
    public static void main(String[] args) {
        int i = 10;
        int j = --i + ++i + i + i++ + i--;
        //      9    10    10  10    11
        System.out.println(i); //10
        System.out.println(j); //9+10+10+10+11=50
    }
}
```

```
<terminated> B (
10
50
```

18/05/2022 (Wednesday)

PACKAGES IN JAVA

Packages in Java are nothing but folders created to store the programs in an organized manner.

- A Package name cannot be a keyword.
- Don't give package name in UPPERCASE letters, encouraged to be given in lowercase letters.
- If we want to use a class present in the same package then importing is not required but if we are using a class that is present indifferent package then importing that becomes mandatory.

Example:

C.java
```
package p2;
import p1.A; //we need to import if extending class from other package
             //import packageName.ClassName
             //maintain the same hierarchy of importing
public class C extends A {

}
```

*B.java
```
package p1;
    //extending the class from same package
    //no need to import
public class B extends A {

}
```

A.java
```
package p1;

public class A {

}
```

When we create an object of "Class A" in "Class B" then they're called as "Non-Sub Class", when we extend a class in same package then it's called as "Sub-Class".

Example: *(non-sub class)*

A.java
```
package p1;

public class A {

}
```

*B.java
```
package p1;
    //non sub class
public class B {
    public static void main(String[] args) {
        A a1 = new A();
    }

}
```

Example #2: *(non-sub class)*

A.java
```
package p1;

public class A {

}
```

B.java
```
package p1;
    //non sub class
public class B {
    public static void main(String[] args)
        A a1 = new A();
    }

}
```

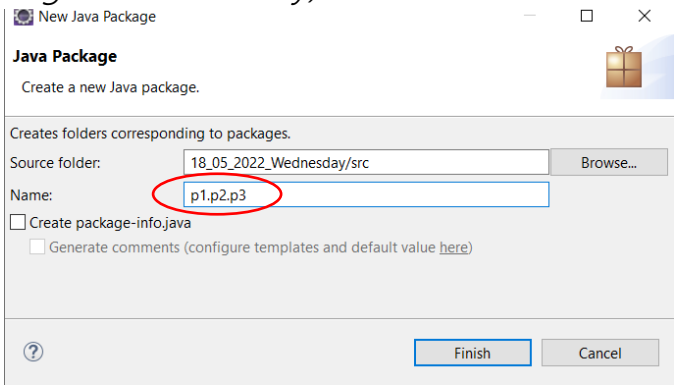
C.java
```
package p2;
    //non sub class
import p1.A;
public class C {
    public static void main(String[] args)
        A a1 = new A();
    }
}
```

Example: *(sub class)*

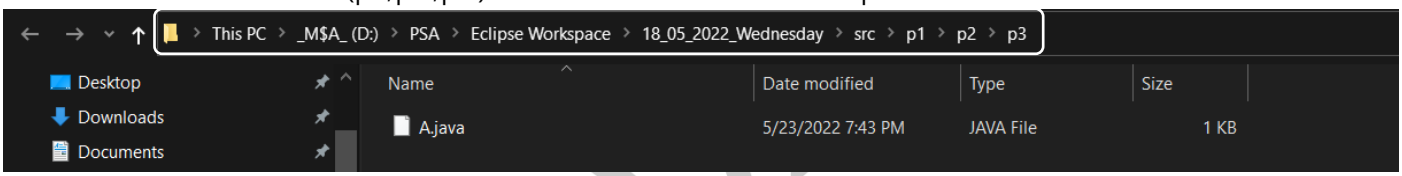
A.java
```
1 package p1;
2
3 public class A {
4
5 }
6
```

*B.java
```
1 package p1;
2     //sub class
3 public class B extends A {
4     public static void main(String[] args) {
5         A a1 = new A();
6     }
7
8 }
9
```

Example: *(Creating a package in another way)*



It creates three folders (p1,p2,p3) & class will be saved in "p3" folder.



B.java
```
1 package p4;
2 import p1.p2.p3.A;
3 public class B extends A {
4
5 }
6
```

A.java
```
1 package p1.p2.p3;
2
3 public class A {
4
5 }
6
```

Without using "import" keyword, importing can be done with *(packageName.ClassName)*, see the below Example:

*B.java
```
1 package p2;
2 //did not import p1, but importing done with packageName.ClassName
3 public class B extends p1.A {
4     public static void main(String[] args) {
5         p1.A a1 = new p1.A();
6     }
7
8 }
9
```

A.java
```
1 package p1;
2
3 public class A {
4
5 }
6
```

Packages resolves naming convention problem in JAVA, that is we can create more than one class in the same project provided packages are different as shown in the below example:

*B.java
```
1 package p3;
2
3 public class B {
4     public static void main(String[] args) {
5         //p1
6         p1.A a1 = new p1.A();
7         //p2
8         p2.A a2 = new p2.A();
9     }
10 }
11
```

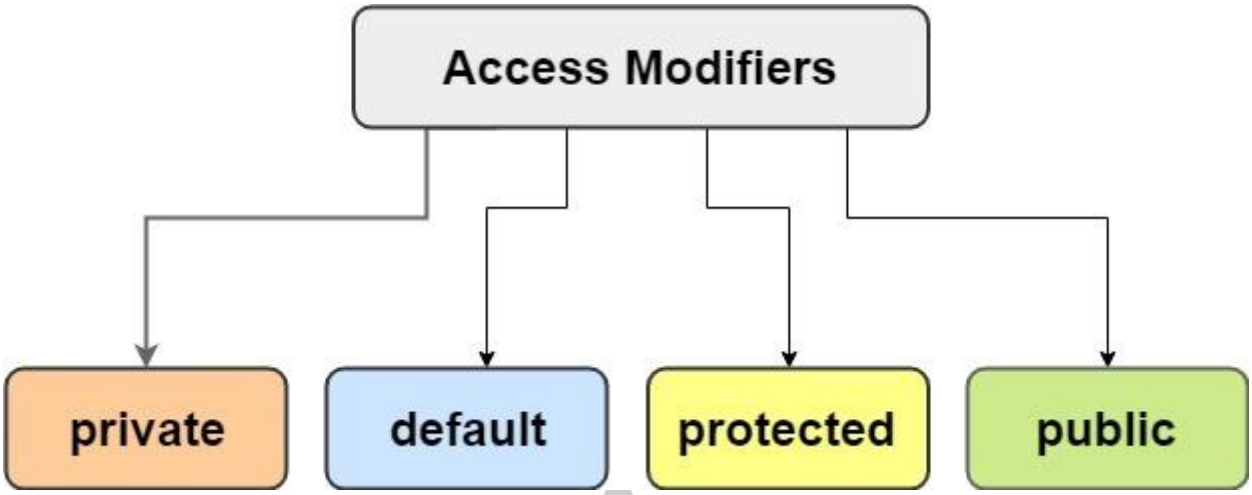
A.java
```
1 package p2;
2
3 public class A {
4
5 }
6
```

A.java
```
1 package p1;
2
3 public class A {
4
5 }
6
```

19/05/2022 (Thursday)

ACCESS SPECIFIERS/MODIFIERS IN JAVA

There are four types of access specifiers/modifiers in JAVA.



Access Specifier	Private	Default	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package (Sub Class)	No	Yes	Yes	Yes
Same Package (Non SubClass)	No	Yes	Yes	Yes
Different Package (SubClass)	No	No	Yes	Yes
Different Package (Non SubClass)	No	No	No	Yes

Examples on Private: Access Specifier

- If we make variable/method private, then it can be accessed only in same class but not outside class.

Example #1: *(same class)*

*A.java

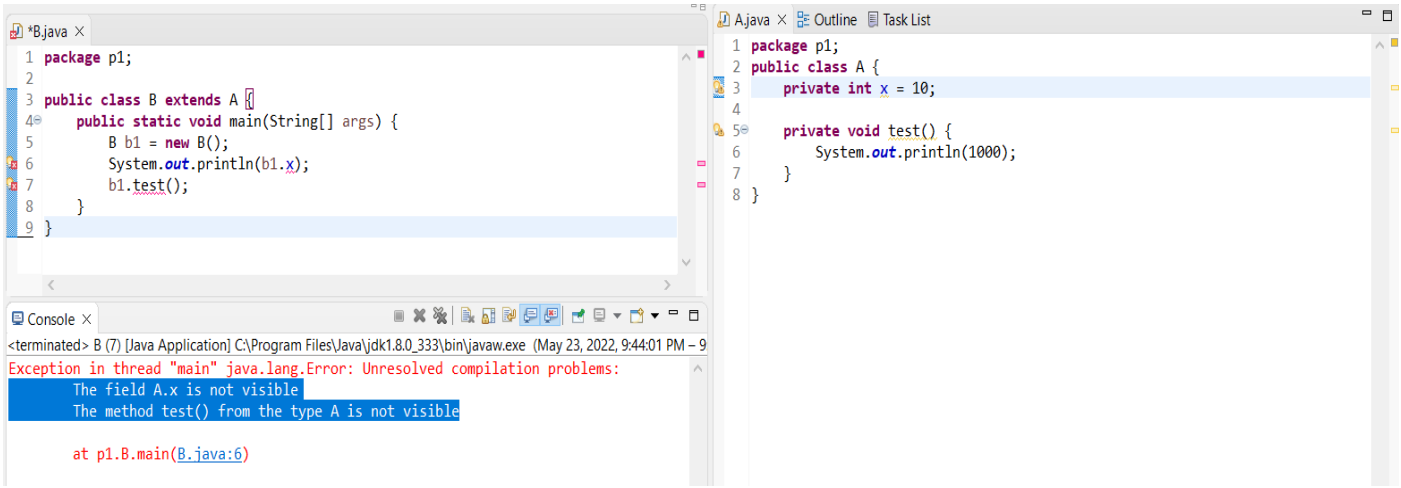
19_05_2022_Thursday/src/p1/A.java

```
1
2  public class A {
3      private int x = 10;
4
5      private void test() {
6          System.out.println(1000);
7      }
8      public static void main(String[] args) {
9          A a1 = new A();
10         System.out.println(a1.x);
11         a1.test();
12     }
13 }
```

Console

```
<terminated> A (14)
10
1000
```

Example #2: *(same package, sub class)*



*B.java

```
1 package p1;
2
3 public class B extends A {
4     public static void main(String[] args) {
5         B b1 = new B();
6         System.out.println(b1.x);
7         b1.test();
8     }
9 }
```

A.java

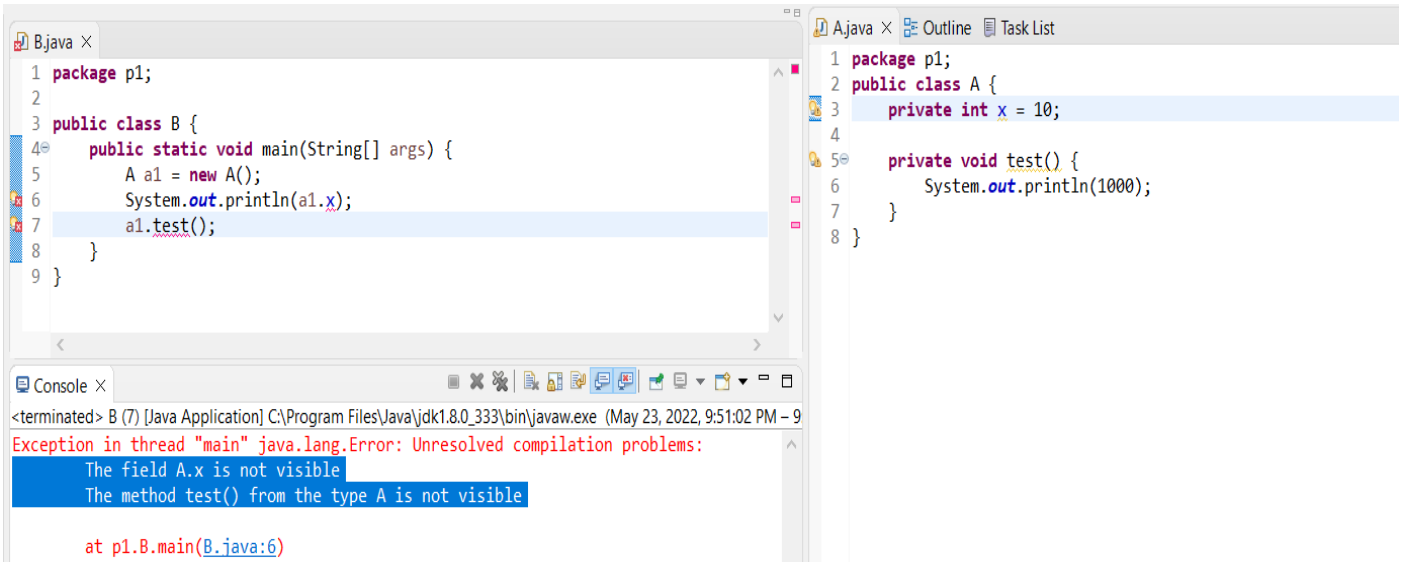
```
1 package p1;
2 public class A {
3     private int x = 10;
4
5     private void test() {
6         System.out.println(1000);
7     }
8 }
```

Console

```
<terminated> B (7) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2022, 9:44:01 PM – 9
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field A.x is not visible
        The method test() from the type A is not visible

        at p1.B.main(B.java:6)
```

Example #3: *(same package, non sub class)*



B.java

```
1 package p1;
2
3 public class B {
4     public static void main(String[] args) {
5         A a1 = new A();
6         System.out.println(a1.x);
7         a1.test();
8     }
9 }
```

A.java

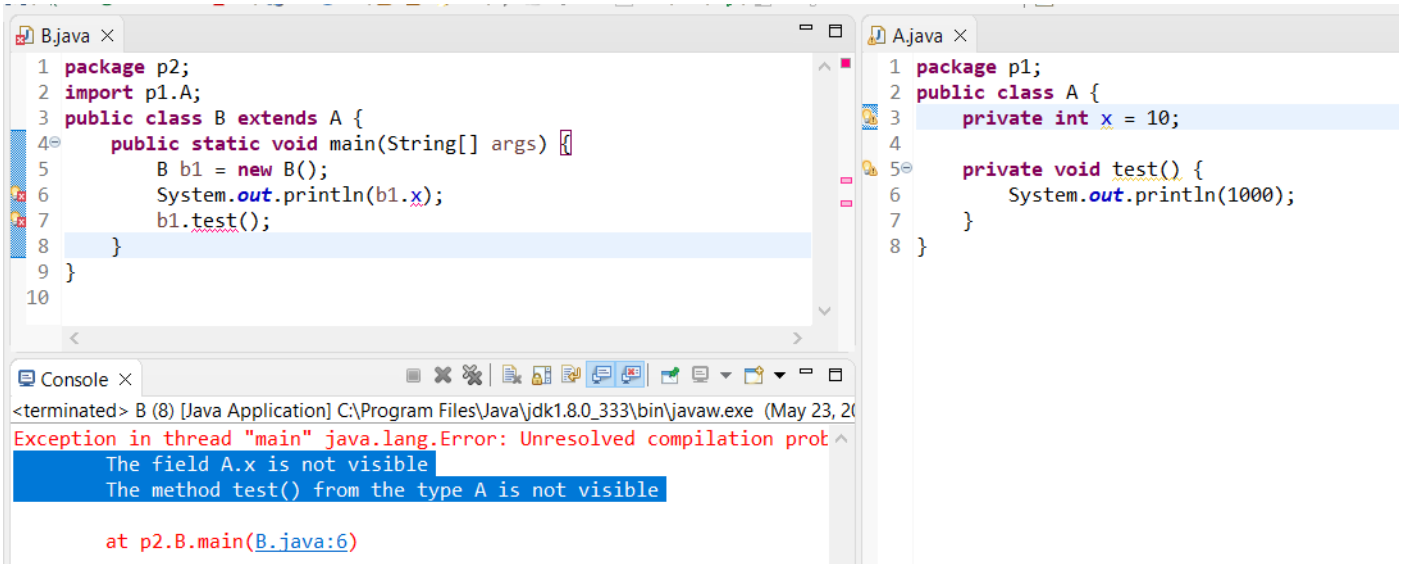
```
1 package p1;
2 public class A {
3     private int x = 10;
4
5     private void test() {
6         System.out.println(1000);
7     }
8 }
```

Console

```
<terminated> B (7) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2022, 9:51:02 PM – 9
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field A.x is not visible
        The method test() from the type A is not visible

        at p1.B.main(B.java:6)
```

Example #4: *(different package, sub class)*



B.java

```
1 package p2;
2 import p1.A;
3 public class B extends A {
4     public static void main(String[] args) {
5         B b1 = new B();
6         System.out.println(b1.x);
7         b1.test();
8     }
9 }
10
```

A.java

```
1 package p1;
2 public class A {
3     private int x = 10;
4
5     private void test() {
6         System.out.println(1000);
7     }
8 }
```

Console

```
<terminated> B (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2
Exception in thread "main" java.lang.Error: Unresolved compilation prob
        The field A.x is not visible
        The method test() from the type A is not visible

        at p2.B.main(B.java:6)
```

Example #5: *(different package, non sub class)*

B.java

```
1  package p2;
2  import p1.A;
3  public class B {
4      public static void main(String[] args) {
5          A a1 = new A();
6          System.out.println(a1.x);
7          a1.test();
8      }
9  }
10
```

A.java

```
1  package p1;
2  public class A {
3      private int x = 10;
4
5      private void test() {
6          System.out.println(1000);
7      }
8  }
```

Console

```
<terminated> B (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 23, 2(
Exception in thread "main" java.lang.Error: Unresolved compilation prob
        The field A.x is not visible
        The method test() from the type A is not visible

        at p2.B.main(B.java:6)
```

Examples on Default: Access Specifier

- If we make variable/method default, then it can be accessed in same class, same package but not in different packages.

Example #1: *(same class)*

A.java

```
1  package p1;
2  public class A {
3      int x = 10; //Access specifier = default (not writing anything befoe variable makes it default)
4      void test() {
5          System.out.println(1000);
6      }
7      public static void main(String[] args) {
8          A a1 = new A();
9          System.out.println(a1.x);
10         a1.test();
11     }
12 }
```

Console

```
<terminated> A
10
1000
```

Example #2: *(same package, sub class)*

B.java

```
1  package p1;
2
3  public class B extends A{
4      public static void main(String[] args) {
5          B b1 = new B();
6          System.out.println(b1.x);
7          b1.test();
8      }
9  }
```

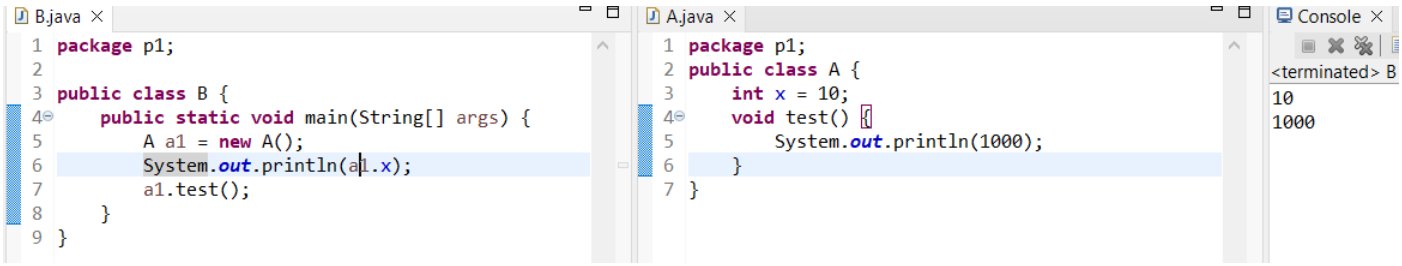
A.java

```
1  package p1;
2  public class A {
3      int x = 10;
4      void test() {
5          System.out.println(1000);
6      }
7  }
```

Console

```
<terminated> B
10
1000
```

Example #3: *(same package, non sub class)*



B.java

```
1 package p1;
2
3 public class B {
4     public static void main(String[] args) {
5         A a1 = new A();
6         System.out.println(a1.x);
7         a1.test();
8     }
9 }
```

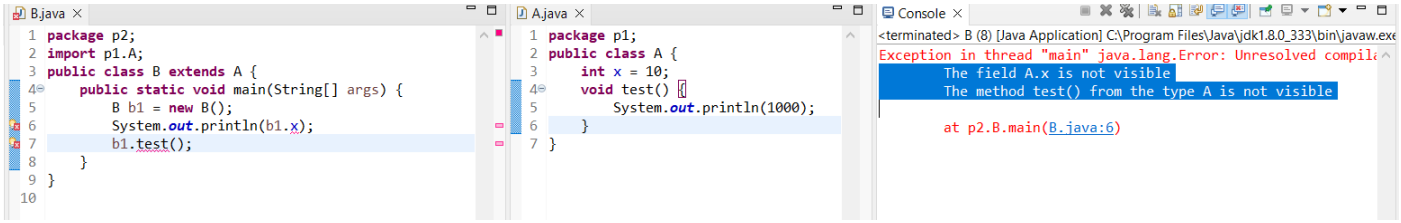
A.java

```
1 package p1;
2 public class A {
3     int x = 10;
4     void test() {
5         System.out.println(1000);
6     }
7 }
```

Console

```
<terminated> B
10
1000
```

Example #4: *(different package, sub class)*



B.java

```
1 package p2;
2 import p1.A;
3 public class B extends A {
4     public static void main(String[] args) {
5         B b1 = new B();
6         System.out.println(b1.x);
7         b1.test();
8     }
9 }
10
```

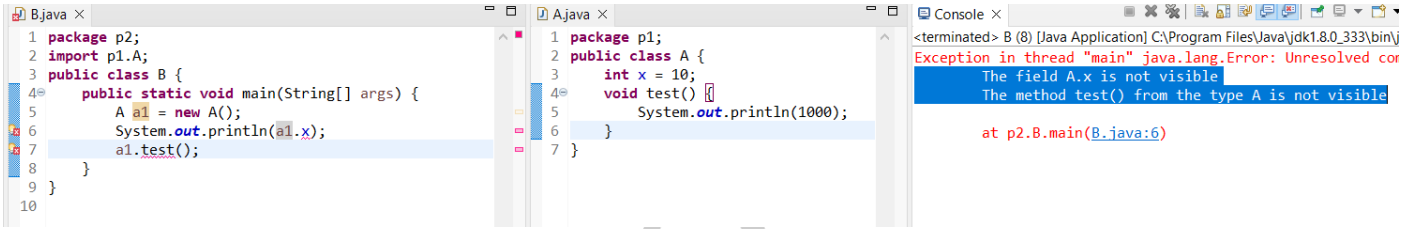
A.java

```
1 package p1;
2 public class A {
3     int x = 10;
4     void test() {
5         System.out.println(1000);
6     }
7 }
```

Console

```
<terminated> B (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe
Exception in thread "main" java.lang.Error: Unresolved compila
        The field A.x is not visible
        The method test() from the type A is not visible
        at p2.B.main(B.java:6)
```

Example #5: *(different package, non sub class)*



B.java

```
1 package p2;
2 import p1.A;
3 public class B {
4     public static void main(String[] args) {
5         A a1 = new A();
6         System.out.println(a1.x);
7         a1.test();
8     }
9 }
10
```

A.java

```
1 package p1;
2 public class A {
3     int x = 10;
4     void test() {
5         System.out.println(1000);
6     }
7 }
```

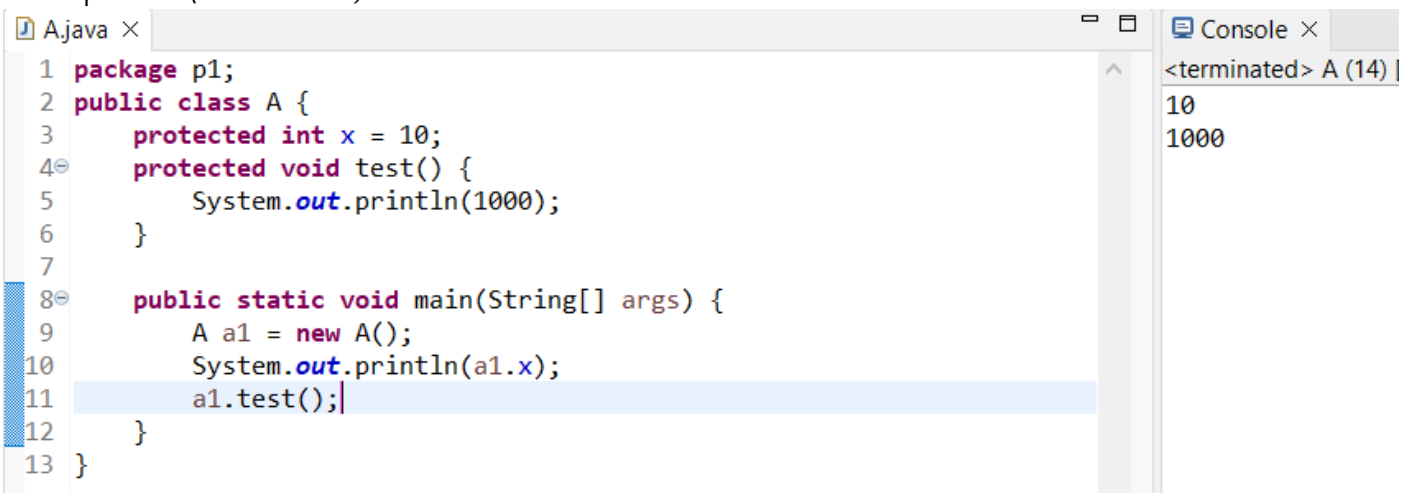
Console

```
<terminated> B (8) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\j
Exception in thread "main" java.lang.Error: Unresolved cor
        The field A.x is not visible
        The method test() from the type A is not visible
        at p2.B.main(B.java:6)
```

Examples on Protected: Access Specifier

- If we make variable/method protected, then it can be accessed in same class, same package & in different packages only via inheritance.

Example #1: *(same class)*



A.java

```
1  package p1;
2  public class A {
3      protected int x = 10;
4      protected void test() {
5          System.out.println(1000);
6      }
7
8      public static void main(String[] args) {
9          A a1 = new A();
10         System.out.println(a1.x);
11         a1.test();
12     }
13 }
```

Console

```
<terminated> A (14)
10
1000
```

Example #2: *(same package, sub class)*

B.java
```
1  package p1;
2
3  public class B extends A {
4      public static void main(String[] args) {
5          B b1 = new B();
6          System.out.println(b1.x);
7          b1.test();
8      }
9  }
```

A.java
```
1  package p1;
2  public class A {
3      protected int x = 10;
4      protected void test() {
5          System.out.println(1000);
6      }
7  }
```

Console
```
<terminated> B
10
1000
```

Example #3: *(same package, non sub class)*

B.java
```
1  package p1;
2
3  public class B {
4      public static void main(String[] args) {
5          A a1 = new A();
6          System.out.println(a1.x);
7          a1.test();
8      }
9  }
```

A.java
```
1  package p1;
2  public class A {
3      protected int x = 10;
4      protected void test() {
5          System.out.println(1000);
6      }
7  }
```

Console
```
<terminated> B
10
1000
```

Example #4: *(different package, sub class)*

B.java
```
1  package p2;
2  import p1.A;
3  public class B extends A {
4      public static void main(String[] args) {
5          B b1 = new B();
6          System.out.println(b1.x);
7          b1.test();
8      }
9  }
10
```

A.java
```
1  package p1;
2  public class A {
3      protected int x = 10;
4      protected void test() {
5          System.out.println(1000);
6      }
7  }
```

Console
```
<terminated> B
10
1000
```

Example #5: *(different package, non sub class)*

B.java
```
1  package p2;
2  import p1.A;
3  public class B {
4      public static void main(String[] args) {
5          A a1 = new A();
6          System.out.println(a1.x);
7          a1.test();
8      }
9  }
10
```

A.java
```
1  package p1;
2  public class A {
3      protected int x = 10;
4      protected void test() {
5          System.out.println(1000);
6      }
7  }
```

Console
```
<terminated> B (8) [Java Application] C:\Program Files\Java\jdk1.8.0_3
Exception in thread "main" java.lang.Error: Unresolv
        The field A.x is not visible
        The method test() from the type A is not vis

        at p2.B.main(B.java:6)
```

Examples on Public: Access Specifier

- If we make variable/method public, then it can be accessed everywhere.

Example #1: *(same class)*

A.java

```
1  package p1;
2  public class A {
3      public int x = 10;
4      public void test() {
5          System.out.println(1000);
6      }
7      public static void main(String[] args) {
8          A a1 = new A();
9          System.out.println(a1.x);
10         a1.test();
11     }
12 }
```

Console

```
<terminated> A
10
1000
```

Example #2: *(same package, sub class)*

B.java

```
1  package p1;
2
3  public class B extends A {
4      public static void main(String[] args) {
5          B b1 = new B();
6          System.out.println(b1.x);
7          b1.test();
8      }
9  }
```

A.java

```
1  package p1;
2  public class A {
3      public int x = 10;
4      public void test() {
5          System.out.println(1000);
6      }
7  }
```

Console

```
<terminated> B
10
1000
```

Example #3: *(same package, non sub class)*

B.java

```
1  package p1;
2
3  public class B {
4      public static void main(String[] args) {
5          A a1 = new A();
6          System.out.println(a1.x);
7          a1.test();
8      }
9  }
```

A.java

```
1  package p1;
2  public class A {
3      public int x = 10;
4      public void test() {
5          System.out.println(1000);
6      }
7  }
```

Console

```
<terminated> B
10
1000
```

Example #4: *(different package, sub class)*

*B.java

```
1  package p2;
2  import p1.A;
3  public class B extends A {
4      public static void main(String[] args) {
5          B b1 = new B();
6          System.out.println(b1.x);
7          b1.test();
8      }
9  }
```

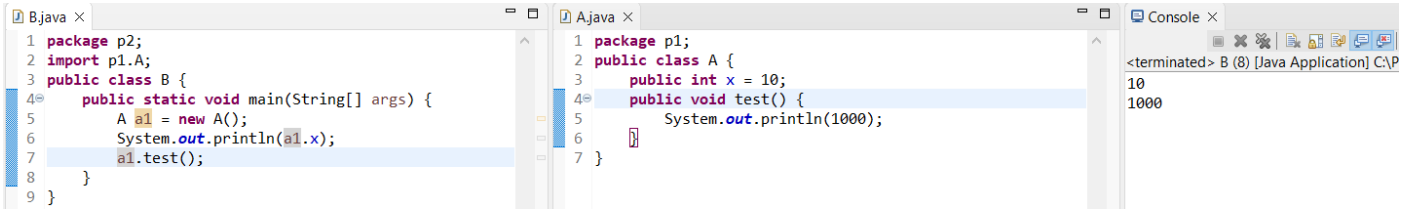
A.java

```
1  package p1;
2  public class A {
3      public int x = 10;
4      public void test() {
5          System.out.println(1000);
6      }
7  }
```

Console

```
<terminated> B
10
1000
```

Example #5: *(different package, non sub class)*



```
B.java
1 package p2;
2 import p1.A;
3 public class B {
4     public static void main(String[] args) {
5         A a1 = new A();
6         System.out.println(a1.x);
7         a1.test();
8     }
9 }
```

```
A.java
1 package p1;
2 public class A {
3     public int x = 10;
4     public void test() {
5         System.out.println(1000);
6     }
7 }
```

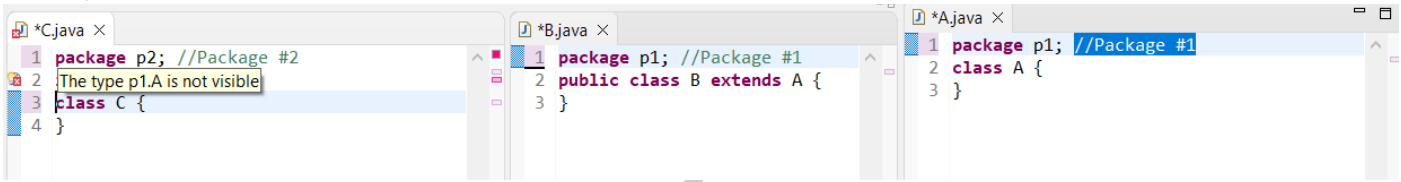
```
Console
<terminated> B (8) [Java Application] C:\P
10
1000
```

Access Specifier on Class

A Class can never be private nor be protected, it can only be default and public.

If we make the Class default, then it can be accessed only in the same package and if we make a class public then it can be accessed everywhere in the program.

Example: *(default)*

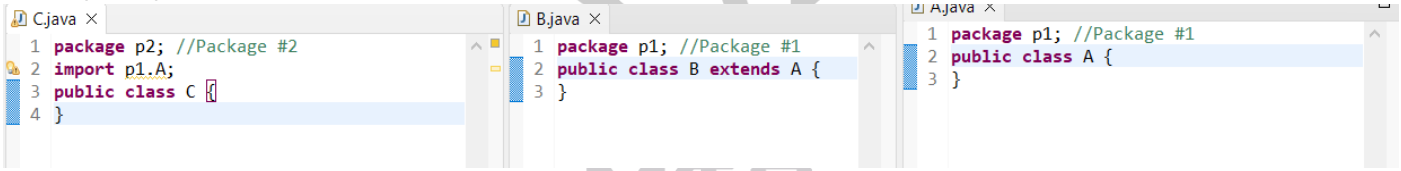


```
*C.java
1 package p2; //Package #2
2 import p1.A;
3 public class C {
4 }
```

```
*B.java
1 package p1; //Package #1
2 public class B extends A {
3 }
```

```
*A.java
1 package p1; //Package #1
2 class A {
3 }
```

Example: *(public)*



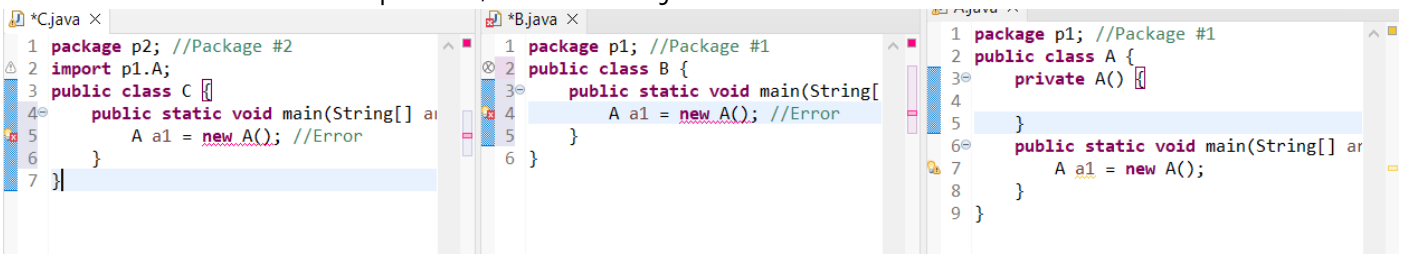
```
C.java
1 package p2; //Package #2
2 import p1.A;
3 public class C {
4 }
```

```
B.java
1 package p1; //Package #1
2 public class B extends A {
3 }
```

```
A.java
1 package p1; //Package #1
2 public class A {
3 }
```

Access Specifier on Constructors

If we make a constructor private, then it's object cannot be created outside the Class. Ex:

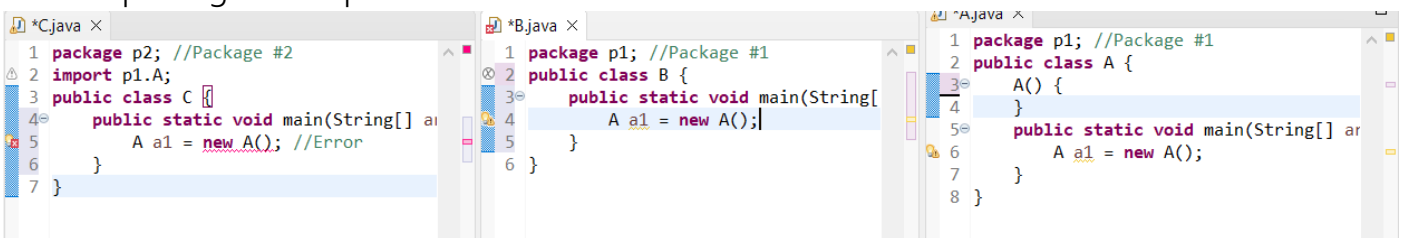


```
*C.java
1 package p2; //Package #2
2 import p1.A;
3 public class C {
4     public static void main(String[] ar
5         A a1 = new A(); //Error
6     }
7 }
```

```
*B.java
1 package p1; //Package #1
2 public class B {
3     public static void main(String[
4         A a1 = new A(); //Error
5     }
6 }
```

```
1 package p1; //Package #1
2 public class A {
3     private A() {
4
5     }
6     public static void main(String[] ar
7         A a1 = new A();
8     }
9 }
```

If we make a constructor default, then it's object can be created in same package but not outside package. Example:



```
*C.java
1 package p2; //Package #2
2 import p1.A;
3 public class C {
4     public static void main(String[] ar
5         A a1 = new A(); //Error
6     }
7 }
```

```
*B.java
1 package p1; //Package #1
2 public class B {
3     public static void main(String[
4         A a1 = new A();
5     }
6 }
```

```
*A.java
1 package p1; //Package #1
2 public class A {
3     A() {
4     }
5     public static void main(String[] ar
6         A a1 = new A();
7     }
8 }
```

If we make a constructor protected, then it's object can be created in same package but not in different package, i.e same as default. Example:

*C.java

```
1  package p2; //Package #2
2  import p1.A;
3  public class C {
4      public static void main(String[] ar
5          A a1 = new A(); //Error
6      }
7  }
```

*B.java

```
1  package p1; //Package #1
2  public class B {
3      public static void main(String[
4          A a1 = new A();
5      }
6  }
```

*A.java

```
1  package p1; //Package #1
2  public class A {
3      A() {
4      }
5      public static void main(String[] ar
6          A a1 = new A();
7      }
8  }
```

If we make a constructor public, then it's object can be created in same package and in different package as well. Example:

*C.java

```
1  package p2; //Package #2
2  import p1.A;
3  public class C {
4      public static void main(String[] ar
5          A a1 = new A(); //Error
6      }
7  }
```

*B.java

```
1  package p1; //Package #1
2  public class B {
3      public static void main(String[
4          A a1 = new A();
5      }
6  }
```

*A.java

```
1  package p1; //Package #1
2  public class A {
3      public A() {
4      }
5      public static void main(String[] ar
6          A a1 = new A();
7      }
8  }
```

Q. What is data hiding?

A. Here we make the variable private so that this variable cannot be accessed outside the class. Data hiding is applicable only on variables and not on methods.

Assignment-2

Watch these YouTube Videos by **Pankaj Sir Academy** Chanel.
Click to Watch:

YouTube IIB: Instance Initialization Block

YouTube SIB: Static Initialization Block

21/05/2022 (Saturday) Assignment-2

INSTANCE INITIALIZATION BLOCK (IIB)

IIBs are executed when objects are created. No. of times we create the object, same no. of times IIB will be called. IIBs are used to initialize all the instance (non static) variables in one place, and that gives us better readability of the code.

Example #1: *(No Output, because no object is created, IIB will not be called)*

A.java

```
1  package p1;
2
3  public class A {
4      {
5          System.out.println("From IIB");
6      }
7      public static void main(String[] args) {
8          //No Object is created
9      }
10 }
```

Console

```
<terminated> A (21) [Java Application] C:\P
```

Example #2: *(Object Created)*

*A.java

```
1  package p1;
2
3  public class A {
4
5      {
6          System.out.println("From IIB");
7      }
8
9      public static void main(String[] args) {
10         A a1 = new A(); //Object Created
11     }
12 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
```

Example #3: *(Object Created twice, no. of time we create object same no. of times IIB will be called)*

A.java

```
1  package p1;
2
3  public class A {
4
5      {
6          System.out.println("From IIB");
7      }
8
9      public static void main(String[] args) {
10         A a1 = new A();
11         A a2 = new A();
12     }
13 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From IIB
```

Example #4: *(Creating one constructor with IIB & see which is called first)*

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      A() {   //Constructor
5          System.out.println("From Constructor");
6      }
7
8      {    //IIB
9          System.out.println("From IIB");
10     }
11
12     public static void main(String[] args) {
13         A a1 = new A();
14     }
15 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From Constructor
```

Example #5:

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      A() {   //Constructor
5          System.out.println("From Constructor");
6      }
7
8      {    //IIB
9          System.out.println("From IIB");
10     }
11
12     public static void main(String[] args) {
13         A a1 = new A();
14         System.out.println("From Main");
15     }
16 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From Constructor
From Main
```

Example #6:

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      A() {   //Constructor
5          System.out.println("From Constructor");
6      }
7
8      {    //IIB
9          System.out.println("From IIB");
10     }
11
12     public static void main(String[] args) {
13         System.out.println("Start Main");
14         A a1 = new A();
15         System.out.println("End Main");
16     }
17 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
Start Main
From IIB
From Constructor
End Main
```

Example #6: (Multiple IIBs)

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      {    //IIB-1
5          System.out.println("From IIB");
6      }
7
8      A() {   //Constructor
9          System.out.println("From Constructor");
10     }
11
12     {    //IIB-2
13         System.out.println("From IIB");
14     }
15
16     public static void main(String[] args) {
17         A a1 = new A();
18     }
19 }
```

Console

```
<terminated> A (21) [Java Application] C:\P
From IIB
From IIB
From Constructor
```

Example #7:

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      int i;
5
6      {
7          i=100;
8          System.out.println(i);
9      }
10
11     public static void main(String[] args) {
12         A a1 = new A();
13     }
14 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
100
```

We can initialize both static and non static variables inside an IIB.

Example #8:

A.java

```
1  package p1;
2      //Instance Initialization Block (IIB)
3  public class A {
4      static int i;
5
6      {
7          i=100;
8          System.out.println(i);
9      }
10
11     public static void main(String[] args) {
12         A a1 = new A();
13     }
14 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
100
```

STATIC INITIALIZATION BLOCK (IIB)

Static Initialization Block (SIB) are created inside the class with the “static” keyword & executed before main method, it runs only once, we need not create an object for SIB.

Example:

*A.java

```
1  package p1;
2      //Static Initialization Block (IIB)
3  public class A {
4
5      static
6      {
7          System.out.println("From SIB");
8      }
9
10     public static void main(String[] args) {
11         //No Object Created
12     }
13 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pro
From SIB
```

Example #2:

A.java

```
1  package p1;
2      //Static Initialization Block (IIB)
3  public class A {
4
5      static
6      {
7          System.out.println("From SIB");
8      }
9
10     public static void main(String[] args) {
11         System.out.println("From Main");
12     }
13 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB
From Main
```

Example #3:

A.java

```
1  package p1;
2      //Static Initialization Block (IIB)
3  public class A {
4      static int i;
5      static
6      {
7          i=25;
8          System.out.println(i);
9      }
10
11     public static void main(String[] args) {
12         System.out.println("From Main");
13     }
14 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
25
From Main
```

Example #4: *(Multiple SIBs)*

A.java

```
1  package p1;
2      //Static Initialization Block (IIB)
3  public class A {
4      static
5      {
6          System.out.println("From SIB-1");
7      }
8
9      static
10     {
11         System.out.println("From SIB-2");
12     }
13
14     public static void main(String[] args) {
15         System.out.println("From Main");
16     }
17 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB-1
From SIB-2
From Main
```

Example #5: *(SIB can only initialize static variable, it cannot initialize non static variable)*

A.java

```
1  package p1;
2      //Static Initialization Block (IIB)
3  public class A {
4      int i;
5      static
6      {
7          i = 100;
8      }
9
10     public static void main(String[] args) {
11         System.out.println("From Main");
12     }
13 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
Exception in thread "main" java.l

        at p1.A.main(A.java:10)
```

Example: *(IIB and SIB)*

A.java

```
1  package p1;
2      //IIB and SIB
3  public class A {
4      {
5          System.out.println("From IIB");
6      }
7      static
8      {
9          System.out.println("From SIB");
10     }
11     public static void main(String[] args) {
12         A a1 = new A();
13     }
14 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB
From IIB
```

Example: *(To understand the flow between SIB, IIB, Constructor and Main)*

A.java

```
1  package p1;
2      //IIB, SIB and ConstructorD
3  public class A {
4      {
5          System.out.println("From IIB");
6      }
7
8      A() {
9          System.out.println("From Constructor");
10     }
11
12     static
13     {
14         System.out.println("From SIB");
15     }
16     public static void main(String[] args) {
17         A a1 = new A();
18         System.out.println("From Main");
19     }
20 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB
From IIB
From Constructor
From Main
```

Example:

A.java

```
1  package p1;
2      //IIB, SIB and ConstructorD
3  public class A {
4      {
5          System.out.println("From IIB-1");
6      }
7
8      {
9          System.out.println("From IIB-2");
10     }
11
12     static
13     {
14         System.out.println("From SIB");
15     }
16
17     A() {
18         System.out.println("From Constructor");
19     }
20
21
22     public static void main(String[] args) {
23         A a1 = new A();
24         System.out.println("From Main");
25     }
26 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB
From IIB-1
From IIB-2
From Constructor
From Main
```

Example: *(Object created in SIB)*

A.java

```
1  package p1;
2      //IIB, SIB and Constructor Flow
3  public class A {
4      {
5          System.out.println("From IIB");
6      }
7
8      static {
9          new A(); //Object created in SIB, so due to this IIB runs then constructor called
10         System.out.println("From SIB");
11     }
12
13     A() {
14         System.out.println("From Constructor");
15     }
16
17     public static void main(String[] args) {
18         System.out.println("From Main");
19     }
20 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From Constructor
From SIB
From Main
```

Example: *(multiple objects created, in SIB and Main)*

A.java

```
1  package p1;
2      //IIB, SIB and Constructor Flow
3  public class A {
4      static {
5          System.out.println("From SIB-1");
6      }
7
8      static {
9          System.out.println("From SIB-2");
10         new A();
11     }
12
13     {
14         System.out.println("From IIB");
15     }
16
17     public static void main(String[] args) {
18         new A();
19     }
20 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From SIB-1
From SIB-2
From IIB
From IIB
```

Example: *(With test method, IIB and Constructor after main method)*

A.java

```
1  package p1;
2
3  public class A {
4      public void test() {
5          System.out.println("From Test");
6      }
7
8      public static void main(String[] args) {
9          new A().test();
10     }
11
12     { //IIB
13         System.out.println("From IIB");
14     }
15
16     A() { //Constructor
17             System.out.println("From Constructor");
18     }
19 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From Constructor
From Test
```

If we create an object inside IIB, we'll not get any error but then the program will halt abruptly. Example:

A.java

```
1  package p1;
2
3  public class A {
4      {
5          System.out.println("From IIB");
6          new A();
7      }
8
9      public static void main(String[] args) {
10         A a1 = new A();
11         System.out.println("From Main");
12     }
13 }
```

Console

```
<terminated> A (21) [Java Application] C:\Program Files\J
From IIB
From IIB
From IIB
From IIB
java.lang.StackOverflowError
        at java.io.FileOutputStream.write
        at java.io.BufferedOutputStream.
        at java.io.BufferedOutputStream.
        at java.io.PrintStream.write(Pri
        at sun.nio.cs.StreamEncoder.write
        at sun.nio.cs.StreamEncoder.impl
        at sun.nio.cs.StreamEncoder.flus
```

Example: *(multiple object in SIB)*

A.java

```
1  package p1;
2
3  public class A {
4      {
5          System.out.println("From IIB");
6      }
7
8      static
9      {
10         new A();
11         System.out.println("From SIB");
12         new A();
13     }
14
15     public static void main(String[] args) {
16         System.out.println("From Main");
17     }
18 }
```

Console

```
<terminated> A (21) [Java Application] C:\Pr
From IIB
From SIB
From IIB
From Main
```

20/05/2022 (Friday)

OOPs CONCEPTS (contd...)

Polymorphism

If we develop the feature such that it can take more than one form depending upon the situation.

Note: Polymorphism is applicable only on methods and not on variables.

There are two ways we can achieve polymorphism, they are:

i. Overriding
ii. Overloading

i. Overriding: here we inherit a method from the parent class and then we modify its logic in the child class by once again creating a method in child class with same signature. Advantage of overriding is, we can achieve modification of inherited method's logic. Example:

*B.java

```
1  package p1;
2  //Polymorphism
3  public class B extends A{
4      public void test() { //here test method is taking more than one form
5          System.out.println(500); //Overriding 500 with 100 in Class A
6      }
7      public static void main(String[] args) {
8          B b1 = new B();
9          b1.test();
10     }
11 }
12
```

A.java

```
1  package p1;
2
3  public class A {
4      public void test() {
5          System.out.println(100);
6      }
7
8  }
9
```

Console

```
<terminated> B
500
```

Example #2:

B.java

```
1  package p1;
2  //Polymorphism
3  public class B extends A{
4      public void test() { //Here test method is taking more than one form
5          System.out.println(500); //Overriding 500 with 100 in Class A
6      }
7      public static void main(String[] args) {
8          B b1 = new B();
9          b1.test();
10
11         A a1 = new A(); //we created object of the parent class, so it printed 100
12         a1.test();
13     }
14 }
15
```

A.java

```
1  package p1;
2
3  public class A {
4      public void test() {
5          System.out.println(100);
6      }
7
8  }
9
```

Console

```
<terminated> B (
500
100
```

Example #3:

B.java

```
1  package p1;
2  //Polymorphism
3  public class B extends A{
4      public void test2() {
5          System.out.println(500); //Overriding only test2's logic
6      }
7      public static void main(String[] args) {
8          B b1 = new B();
9          b1.test1();
10         b1.test2();
11     }
12 }
13
```

A.java

```
1  package p1;
2
3  public class A {
4      public void test1() {
5          System.out.println(100);
6      }
7      public void test2() {
8          System.out.println(200);
9      }
10 }
11
```

Console

```
<terminated> B
100
500
```

To check whether overriding is happening or not we use "@Override" annotation. This annotation instructs the compiler to report an error if there is a mis-match in signature.

Example:

B.java
```
1  package p1;
2  //Polymorphism
3  public class B extends A{
4      @Override //to check whether override is happening or not
5      public void test() { //Error
6          System.out.println(500);
7      }
8      public static void main(String[] args) {
9          B b1 = new B();
10         b1.test1();
11     }
12 }
13
```

A.java
```
1  package p1;
2
3  public class A {
4      public void test1() {
5          System.out.println(100);
6      }
7  }
8
```

Overriding method with arguments: in argument data type and number of arguments should match, variableName can be anything. Example:

B.java
```
1  package p1;
2  //Polymorphism
3  public class B extends A{
4      @Override
5      public void test1(int y) { //argument: data type should match, varibalename can be anything
6          System.out.println(y);
7      }
8      public static void main(String[] args) {
9          B b1 = new B();
10         b1.test1(800);
11     }
12 }
13
```

A.java
```
1  package p1;
2
3  public class A {
4      public void test1(int x) { //arg
5      System.out.println(x);
6      }
7  }
8
```

<terminated>
800

Example #2:

B.java
```
1  package p1;
2  public class B extends A{
3      @Override
4      public int test() {
5          int x = 1000;
6          return x;
7      }
8      public static void main(String[] args) {
9          B b1 = new B();
10         int val = b1.test();
11         System.out.println(val);
12     }
13 }
14
```

A.java
```
1  package p1;
2
3  public class A {
4      public int test() {
5          int x = 100;
6          return x;
7      }
8  }
9
```

<terminated> B
1000

Example #3:

*PlatinumAccount.java
```
1  package p1;
2
3  public class PlatinumAccount extends GoldAccount {
4
5      @Override
6      public void chqBooks() {
7          System.out.println("Unlimited");
8      }
9
10     @Override
11     public void rateOfInterest() {
12         System.out.println("6% PA");
13     }
14
15     public static void main(String[] args) {
16         GoldAccount g = new GoldAccount();
17             g.onlineBanking();
18             g.rateOfInterest();
19             g.chqBooks();
20
21         PlatinumAccount p =new PlatinumAccount();
22             p.onlineBanking();
23             p.rateOfInterest();
24             p.chqBooks();
25     }
26 }
```

GoldAccount.java
```
1  package p1;
2
3  public class GoldAccount {
4      public void onlineBanking() {
5          System.out.println("Yes");
6      }
7
8      public void chqBooks() {
9          System.out.println("2/Year");
10     }
11
12     public void rateOfInterest() {
13         System.out.println("Nill");
14     }
15
16 }
17
```

<terminated> P
Yes
Nill
2/Year
Yes
6% PA
Unlimited

ii. Overloading: here we create more than one method in the same class with the same name provided they have different no. of arguments or different type of argument. Example:

C.java

```
1  package p1;
2  public class C {
3      public void test() { //Method #1
4          System.out.println(100);
5      }
6      public void test (int x) { //Method #2
7          System.out.println(x);
8      }
9      public void test (String x) { //Method #3
10         System.out.println(x);
11     }
12     public static void main(String[] args) { //Method #4
13         C c1 = new C();
14         c1.test();
15         c1.test(200);
16         c1.test("Pankaj Sir Academy");
17     }
18 }
19
```

Console

```
<terminated> C (3) [Java Application] C:\Program Files\Ja
100
200
Pankaj Sir Academy
```

Q. Can we override static methods?

A. Static members are never inherited in Java.

In the below example during compilation "B.x" is converted into "A.x", and hence it prints "0".

B.java

```
1  package p1;
2  public class B extends A{
3      public static void main(String[] args) {
4          System.out.println(B.x); //A.x during compilation
5      }
6  }
```

A.java

```
1  package p1;
2  public class A {
3      static int x;
4  }
5
```

```
<terminated> B (8)
0
```

23/05/2022 (Monday)

OOPs CONCEPTS (contd...)

Encapsulation

Wrapping of data with the methods that operate on that data is called "Encapsulation", here we avoid direct access to the data by making that variable private. To operate on that data, we create publicly defined "getters & setters".

Example #1:

B.java
```
1  package p1;
2  //Encapsulation
3  public class B { //non sub class
4      public static void main(String[] args) {
5          A a1 = new A();
6          a1.setId(100); // here we saved id=Value.
7          int id = a1.getId(); //pressed ctrl+1
8          System.out.println(id);
9      }
10
11 }
12
```

A.java
```
1  package p1;
2  //Encapsulation
3  public class A {
4      private int id; //non static
5
6      //Below method is "Setter"
7      public void setId(int id) { //here "id" is local variable
8          this.id = id; //with "this" we accessed non static member.
9          //so nonStaticId = localId although variables are same
10     }
11
12     //Below method is "Getter"
13     public int getId() { //because void methods can't return value
14         return id;
15     }
16 }
17 //now we'll set & get id value in different Class
```

```
<terminated> B
100
```

Example #2:

D.java
```
1  package p1;
2  public class D {
3      public static void main(String[] args) {
4          C c1 = new C();
5          c1.setName("Pankaj");
6          String name = c1.getName();
7          System.out.println(name);
8      }
9
10
11 }
12
```

*C.java
```
1  package p1; //Encapsulation
2  //Automaticaly create Setters & Getters
3  public class C {
4      private String name; //Select name & Ctrl+1
5
6      public String getName() {
7          return name;
8      }
9
10     public void setName(String name) {
11         this.name = name;
12     }
13 }
```

```
<terminated> D
Pankaj
```

Example #3: *(multiple variables)*

F.java
```
1  package p1;
2
3  public class F {
4      public static void main(String[] args) {
5          E e1 = new E();
6          e1.setId(672);
7          e1.setName("Pankaj");
8          e1.setCity("Bangalore");
9          e1.setFirstInstallment(25000);
10
11         //direct printing Value
12         System.out.println(e1.getId());
13         System.out.println(e1.getName());
14         System.out.println(e1.getCity());
15         System.out.println(e1.getFirstInstallment());
16     }
17 }
18
```

E.java
```
1  package p1;
2  public class E {
3      private int id;
4      private String name;
5      private String city;
6      private int firstInstallment;
7      //Right Click >Source >Generate Getters & setters...
8
9      public int getId() {
10         return id;
11     }
12     public void setId(int id) {
13         this.id = id;
14     }
15     public String getName() {
16         return name;
17     }
18     public void setName(String name) {
19         this.name = name;
20     }
21     public String getCity() {
22         return city;
23     }
24     public void setCity(String city) {
25         this.city = city;
26     }
27     public int getFirstInstallment() {
28         return firstInstallment;
29     }
30     public void setFirstInstallment(int firstInstallment) {
31         this.firstInstallment = firstInstallment;
32     }
33 }
34
```

```
<terminated> F
672
Pankaj
Bangalore
25000
```

INTERFACE IN JAVA

An interface in Java is a blueprint of a class. It sets a protocol, designing standards etc…
An interface consists of only incomplete/abstract methods. Ex:

*A.java

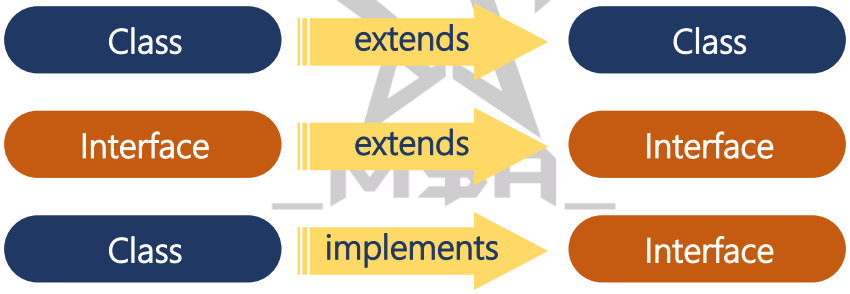
```
1  package p2;
2  public interface A {
3      public void test() { //Error
4
5      }
6      }
7
```

A.java

```
1  package p2;
2  public interface A {
3      public void test(); //Correct
4
5      }
6
```

The relationship between classes and interfaces:

A class extends another class, an interface extends another interface, but a class implements an interface.



Example #1: *(Class-Interface)*

B.java

```
1  package p2;
2
3  public class B implements A { //implemented interface A
4
5      @Override
6      public void test() {
7          System.out.println(1000);
8      }
9
10     public static void main(String[] args) {
11         B b1 = new B();
12         b1.test();
13     }
14 }
```

A.java

```
1  package p2;
2  public interface A {
3      public void test();
4
5      }
6
```

Console

```
<terminated> B
1000
```

Q. What do we achieve by using interface?

A. An interface in Java is a blueprint of a class. It sets a protocol, designing standards etc…

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Example #2: *Logics can be different but methods should be same.*

Bank.java

```
package p2;
//Interface
public interface Bank {
    public void balance();
    public void transfer();

    }

```

HDFCBank.java

```
package p2;
//Company1 doing project for HDFC Bank
public class HDFCBank implements Bank{

    @Override
    public void balance() {
        System.out.println("Logic A");
    }

    @Override
    public void transfer() {
        System.out.println("Logic B");
    }
}
```

SBIBank.java

```
package p2;
//Company2 doing project for SBI Bank
public class SBIBank implements Bank {

    @Override
    public void balance() {
        System.out.println("Logic C");
    }

    @Override
    public void transfer() {
        System.out.println("Logic D");
    }
}
```

Console

```
<terminated> User [Java Application]
Logic A
Logic B
Logic C
Logic D
```

User.java

```
package p2;
//User
public class User {
    public static void main(String[] args) {
        HDFCBank hdfc = new HDFCBank();
        hdfc.balance(); //Logic A
        hdfc.transfer(); //Logic B

        SBIBank sbi = new SBIBank();
        sbi.balance(); //Logic C
        sbi.transfer(); //Logic D
    }
}
```

24/05/2022 (Tuesday)

INTERFACE IN JAVA (Cntd...)

Example #3:

A.java
```
package p1;
public interface A {
    //test1
    public void test1();
}
```

B.java
```
package p1;
public interface B extends A{
    //test1, test2
    public void test2();
}
```

C.java
```
package p1;
public interface C extends B{
    //test1, test2, test3
    public void test3();
}
```

*D.java
```
package p1;
public class D implements C{

    @Override
    public void test2() {
        System.out.println("This was inherited from I2");
    }

    @Override
    public void test1() {
        System.out.println("This was inherited from I1");
    }

    @Override
    public void test3() {
        System.out.println("This was inherited from I3");
    }

    public static void main(String[] args) {
        D d1 = new D();
        d1.test1();
        d1.test2();
        d1.test3();
    }
}
```

Console
```
<terminated> D (2) [Java Application] C:\Program Files\Java\jdk1.8
This was inherited from I1
This was inherited from I2
This was inherited from I3
```

- Interfaces in JAVA supports multiple inheritance, Example:

Example #1:

*A.java
```
package p1;
public interface A {
    public void test1();
}
```

*B.java
```
package p1;
public interface B {
    public void test2();
}
```

C.java
```
package p1;
public interface C extends A,B{
    //multiple inheritance

    public void test3();
}
```

D.java
```
package p1;
public class D implements C{

    @Override
    public void test2() {
        System.out.println("This was inherited from I2");
    }

    @Override
    public void test1() {
        System.out.println("This was inherited from I1");
    }

    @Override
    public void test3() {
        System.out.println("This was inherited from I3");
    }

    public static void main(String[] args) {
        D d1 = new D();
        d1.test1();
        d1.test2();
        d1.test3();
    }
}
```

Console
```
<terminated> D (2) [Java Application] C:\
This was inherited from I1
This was inherited from I2
This was inherited from I3
```

Example #2:

A.java
```
package p1;
public interface A {
    public void test1();
}
```

B.java
```
package p1;
public interface B {
    public void test2();
}
```

C.java
```
package p1;
public interface C {
    public void test3();
}
```

D.java
```
package p1;
public class D implements A,B,C{
    //multiple inheritance

    @Override
    public void test2() {
        System.out.println("This was inherited from IB");
    }

    @Override
    public void test1() {
        System.out.println("This was inherited from IA");
    }

    @Override
    public void test3() {
        System.out.println("This was inherited from IC");
    }

    public static void main(String[] args) {
        D d1 = new D();
        d1.test1();
        d1.test2();
        d1.test3();
    }
}
```

Console
```
<terminated> D (2) [Java Application] C:\
This was inherited from IA
This was inherited from IB
This was inherited from IC
```

Example #3: *(Using extends and implements at the same time, but order should be first extends then implements)*

A.java
```
1 package p1;
2 public interface A {
3     public void test1();
4 }
```

B.java
```
1 package p1;
2 public interface B {
3     public void test2();
4 }
```

C.java
```
1 package p1;
2 public class C {
3     public void test3() {
4         System.out.println("This was inherit
5     }
6 }
```

D.java
```
1  package p1;
2  public class D extends C implements A,B{
3      //using extends and implements both
4
5      @Override
6      public void test2() {
7          System.out.println("This was inherited from IB");
8      }
9
10     @Override
11     public void test1() {
12         System.out.println("This was inherited from IA");
13     }
14
15     public static void main(String[] args) {
16         D d1 = new D();
17         d1.test1();
18         d1.test2();
19         d1.test3();
20     }
21 }
```

Console
```
<terminated> D (2) [Java Application] C:\F
This was inherited from IA
This was inherited from IB
This was inherited from CC
```

Q. Can I create static method in an interface?

A. Static members cannot be inherited.

In an interface incomplete static methods can not be developed hence the below program throws an error. Ex:

*A.java
```
1 package p1;
2 public interface A {
3     public static void test(); //Error
4 }
5
```

B.java
```
1 package p1;
2
3 public class B {
4
5 }
6
```

"final" (keyword):

If we make a variable final then its value once initialized then cannot be changed. Ex:

A.java
```
1 package p1;
2 public class A {
3     public static void main(String[] args) {
4     final int x = 100;
5         x = 10; //Error
6     }
7 }
8
```

Console
```
<terminated> D (2)
Error: Could n
```

If we make static/non static variable final, then it is mandatory to initialize these variables, auto initialization will not happen.

*A.java
```
1 package p1;
2 public class A {
3     final static int x; //Error
4     final int y; //Error
5
6     public static void main(String[] args) {
7     }
8 }
```

If we make a class final then inheritance is not allowed, Ex:

A.java

```
1 package p1;
2 final public class A {
3
4 }
```

*B.java

```
1 package p1;
2 public class B extends A { //Error
3
4 }
5
```

If we make a method final then Overriding is not allowed, Ex:

*A.java

```
1 package p1;
2 public class A {
3     final public void test() {
4
5     }
6 }
```

*B.java

```
1 package p1;
2 public class B extends A {
3
4     @Override
5     public void test() { //Error
6     }
7 }
```

Example: “final” convention (optional): Create variables in UPPERCASE.
Easy way to remember: If a variable is in Blue, *Italics* and Bold, It is *final*.

A.java

```
1 package p1;
2 public class A {
3     final static int MAX_AGE = 75;
4
5 }
```

25/05/2022 (Wednesday)

INTERFACE IN JAVA (Contd...)

Every variable that is created in an interface, by default it is final and static. Ex:

B.java
```
1  package p1;
2  public class B {
3      //cannot inherit static variables,
4      //but accessible by "InterfaceName.variable"
5
6      public static void main(String[] args) {
7          System.out.println(A.MAX_AGE);
8          System.out.println(A.NAME);
9      }
10 }
```

A.java
```
1  package p1;
2  public interface A {
3      int MAX_AGE = 75; //Blue, Italic & Bold: It is final & static.
4      String NAME = "Pankaj";
5  }
```

Console
```
<terminated> B
75
Pankaj
```

Reference variable of an interface can be created, but not object, Ex:

B.java
```
1  package p1;
2  public class B {
3      public static void main(String[] args) {
4          A a1 = new A(); //Error, Cannot instantiate the type A
5      }
6  }
```

A.java
```
1  package p1;
2  public interface A {
3
4  }
```

Console
```
<terminated> B (11) [Java Ap
Exception in thread "m
Cannot instant
at p1.B.main(B
```

UPCASTING IN JAVA

Here we can create parent reference variable and store child class object's address into it and it is termed as "Class Upcasting".

Advantage: Reference variable can store all child class object's addresses.

Example #1: *(Upcasting between Interface & Class)*

A.java
```
1  package p1;
2  public interface A {
3
4  }
```

C.java
```
1  package p1;
2
3  public class C implements A {
4
5  }
6
```

B.java
```
1  package p1;
2  //Class Upcasting
3  public class B implements A {
4      public static void main(String[] args) {
5          A a1 = new B();
6          //Parent Class Ref. var. & child's class obj's address
7          System.out.println(a1);
8
9          a1 =  new C();
10         System.out.println(a1);
11     }
12 }
```

Console
```
<terminated> B (11)
p1.B@15db9742
p1.C@6d06d69c
```

Example #2: *(Upcasting between two Classes)*

A.java
```
1  package p1;
2  public class A {
3
4  }
```

C.java
```
1  package p1;
2
3  public class C extends A {
4
5  }
6
```

B.java
```
1  package p1;
2  //Class Upcasting
3  public class B extends A {
4      public static void main(String[] args) {
5          A a1 = new B();
6          //Parent Class Ref. var. & child's class obj's address
7          System.out.println(a1);
8
9          a1 =  new C();
10         System.out.println(a1);
11     }
12 }
```

Console
```
<terminated> B (11) [Java A
p1.B@15db9742
p1.C@6d06d69c
```

JAVA 8 NEW FEATURES

The following are the JAVA 8 new features:

- Functional Interface
- Lambda Expression
- "default" keyword
- Stream API
- Optional Class

Functional Interface: A functional interface can consist of exactly one incomplete method in it.
Annotation: "@FuctionalInterface"
Example:

A.java

```
1 package p1;
2
3 @FunctionalInterface
4 public interface A {
5     //Error, No Method
6 }
```

(Ex: Error, No Method)

*A.java

```
1 package p1;
2
3 @FunctionalInterface
4 public interface A {
5     //Error, 2 Methods
6     public void test();
7     public void test2();
8 }
```

(Ex: Error, Two Incomplete Methods)

*A.java

```
1 package p1;
2
3 @FunctionalInterface
4 public interface A {
5     //Error, Two Complete Methods
6     public void test() {
7
8     }
9     public void test() {
10
11    }
12 }
```

(Ex: Error, Two complete Methods)

A.java

```
1 package p1;
2
3 @FunctionalInterface
4 public interface A {
5     //No Error, One Incomplete Method
6     public void test();
7 }
```

(Ex: Correct, one Incomplete Method)

Lambda Expression: It reduces the lines of JAVA code, It can only be applied on Functional Interface. In case of lambda expression, we don't need to create an object, implement or Overriding, it does it all automatically. Here, we just write the implementation code. But the drawback is, it makes the code less readable.
Example #1: *(Without using Lambda Expression)*

B.java

```
1  package p1;
2  public class B implements A {
3
4      @Override
5      public void test() {
6          System.out.println(100);
7          System.out.println(200);
8      }
9      public static void main(String[] args) {
10         B b1 = new B();
11         b1.test();
12     }
13 }
```

A.java

```
1 package p1;
2 @FunctionalInterface
3 public interface A {
4     public void test();
5 }
6
```

Console

```
<terminated> B (12
100
200
```

Example #1: *(With Lambda Expression)*

B.java
```
1  package p1;
2  public class B { //No need to write implement
3      public static void main(String[] args) {
4          A a1 = ()-> {
5          System.out.println(100);
6          System.out.println(200);
7          };
8          a1.test();
9      }
10 }
```

A.java
```
1  package p1;
2
3  @FunctionalInterface
4  public interface A {
5      public void test();
6  }
```

Console
```
<terminated> B (
100
200
```

Example #2: *(Method with arguments, Without using Lambda Expression)*

B.java
```
1  package p1;
2  public class B implements A {
3
4      @Override
5      public void test(int a, String b) {
6          System.out.println(a);
7          System.out.println(b);
8      }
9
10     public static void main(String[] args) {
11         B b1 = new B();
12         b1.test(100, "Pankaj");
13     }
14 }
```

A.java
```
1  package p1;
2  @FunctionalInterface
3  public interface A {
4      public void test(int x, String y);
5  }
6
```

Console
```
<terminated> B (1
100
Pankaj
```

Example #2: *(Method with arguments, With Lambda Expression)*

B.java
```
1  package p1;
2  public class B { //No need to write implement
3      public static void main(String[] args) {
4          A a1 = (int a, String b)-> {
5          System.out.println(a);
6          System.out.println(b);
7          };
8          a1.test(100, "Pankaj");
9      }
10 }
```

A.java
```
1  package p1;
2
3  @FunctionalInterface
4  public interface A {
5      public void test(int x, String y);
6  }
```

Console
```
<terminated> B (11) [Java Application]
100
Pankaj
```

Example #3: *(Method with single arguments, Without using Lambda Expression)*

B.java
```
1  package p1;
2  public class B implements A {
3
4      @Override
5      public void test(int a) {
6          System.out.println(a);
7      }
8
9      public static void main(String[] args) {
10         B b1 = new B();
11         b1.test(100);
12     }
13 }
```

A.java
```
1  package p1;
2  @FunctionalInterface
3  public interface A {
4      public void test(int x);
5  }
6
```

Console
```
<terminated> B
100
```

Example #3: *(Method with single arguments, With Lambda Expression & One line code in it)*

B.java
```
1  package p1;
2  public class B { //No need to write implement
3      public static void main(String[] args) {
4          A a1 = (int a)->System.out.println(a);
5          a1.test(100);
6
7      }
8  }
```

A.java
```
1  package p1;
2
3  @FunctionalInterface
4  public interface A {
5      public void test(int x);
6  }
```

Console
```
<terminated> B (
100
```

26/05/2022 (Thursday)

JAVA 8 NEW FEATURES (Contd...)

"default" keyword: "default" keyword was introduced in JDK version 1.8 or JDK version 8. Using "default' keyword we can create complete methods inside an interface.

Note: A functional interface should consist exactly one incomplete method but can have any number of complete methods in it. Example:

B.java

```
1  package p1;
2  public class B implements A {
3
4      @Override
5      public void test1(int a) {
6          System.out.println(a);
7      }
8
9      public static void main(String[] args) {
10         B b1 = new B();
11         b1.test1(100);
12         b1.test2();
13         b1.test3();
14     }
15 }
```

A.java

```
1  package p1;
2  @FunctionalInterface
3  public interface A {
4      public void test1 (int x); //Incomplete Method
5
6      default void test2 () { //Complete Method #1
7          System.out.println(200);
8      }
9
10     default void test3 () { //Complete Method #2
11         System.out.println(300);
12     }
13 }
14
```

Console

```
<terminated> B (12)
100
200
300
```

Example #2: *(Using Lambda Expression with Incomplete & Complete methods in)*

B.java

```
1  package p1;
2  public class B {
3      public static void main(String[] args) {
4          A a1 = (int a)-> { //Lambda Expression
5          System.out.println(a);
6          };
7
8          a1.test1(100);
9          a1.test2();
10         a1.test3();
11
12     }
13 }
```

A.java

```
1  package p1;
2
3  @FunctionalInterface
4  public interface A {
5  public void test1 (int x); //Incomplete Method
6
7      default void test2 () { //Complete Method #1
8          System.out.println(200);
9      }
10
11     default void test3 () { //Complete Method #2
12         System.out.println(300);
13     }
14 }
```

Console

```
<terminated> B (1
100
200
300
```

In an interface we can add main method and can run like a class as shown in the below example, it was not possible in JAVA Version 7:

A.java

```
1  package p1;
2
3  public interface A {
4      int x = 100; //Static & final by default in Interface
5      public static void main(String[] args) {
6          System.out.println(A.x);
7      }
8  }
```

Console

```
<terminated> A (16) [Java Application]
100
```

We can also develop complete static methods in an interface, incomplete and static is not allowed. see the below Example:

A.java

```
1  package p1;
2
3  public interface A {
4      int x = 100; //Static & final by default in Interface
5      public static void main(String[] args) {
6          System.out.println(A.x);
7          A.test();
8      }
9
10     public static void test() { //static & complete method
11         System.out.println(200);
12     }
13 }
```

Console

```
<terminated> A (16) [Java Application]
100
200
```

OOPs CONCEPTS (contd...)

Abstraction (Incompletion)

Q . What is Abstraction in JAVA?

A. Hiding of implementation details is called as "abstraction" & the way we achieve that in JAVA is by using Interfaces and Abstract Class.

"abstract" keyword: It helps us to define incomplete methods & incomplete classes. To develop incomplete methods in interface usage of "abstract" keyword is optional. Example:

A.java

```
1  package p1;
2
3  public interface A {
4      abstract public void test();
5      public void example(); //Automatically abstract added in interface, Optional.
6  }
```

Abstract Class: An abstract class can be 0-100% incomplete, it can consist of a constructor, it can consist of static and non static members, we can create main method in an abstract class, but object of abstract class cannot be created. Example:

A.java

```
1  package p1;
2  //incomplete and complete methods
3  abstract public class A {
4      int x = 10;
5      static int y = 100;
6      A(){ //Constructor
7
8      }
9      abstract public void test1(); //Incomplete Method, Using abstract keyword is mandatory in Class.
10     public void test2 () { //Complete method
11         System.out.println(100);
12     }
13
14     public static void main(String[] args) { //main method
15         A a1 = new A(); //Error
16     }
17 }
```

Console

```
<terminated> A (17) [Java Application] C:\Program Files\Java\jdk1.8.0_
Exception in thread "main" java.lang.Error: Unresolv
        Cannot instantiate the type A

        at p1.A.main(A.java:15)
```

Assignment -3

Watch this YouTube Video by **Pankaj Sir Academy** Chanel, entitled **"super" keyword**.
Click to Watch:

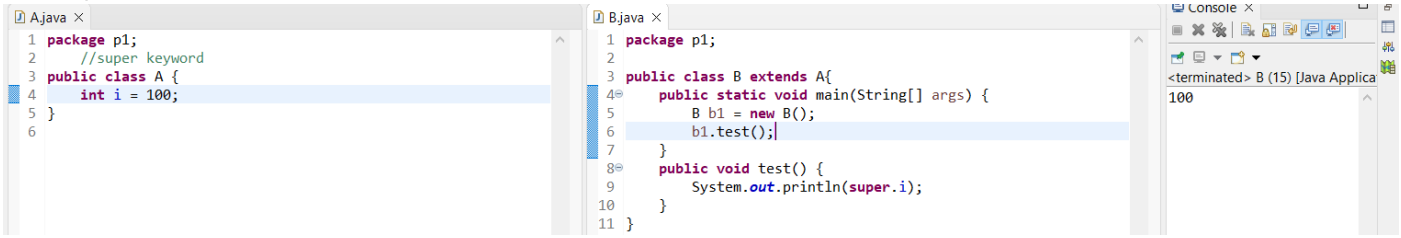
YouTube "super" keyword

27/05/2022 (Friday)

"super" Keyword

Using "super" keyword w can access the members of parent class.

Example #1:

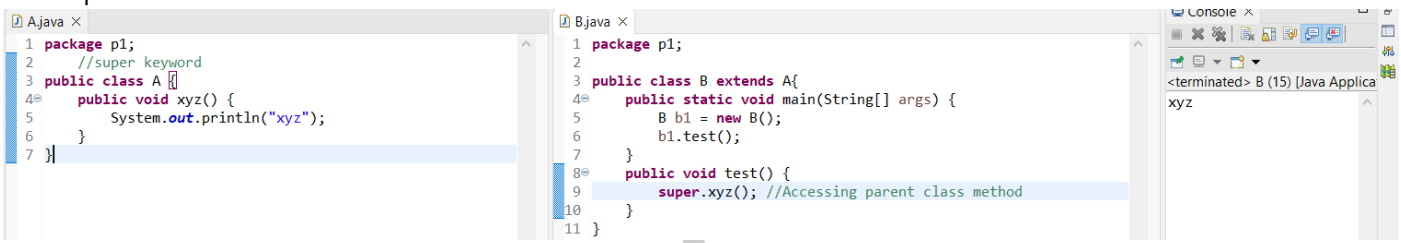


```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     int i = 100;
5 }
6
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     public static void main(String[] args) {
5         B b1 = new B();
6         b1.test();
7     }
8     public void test() {
9         System.out.println(super.i);
10    }
11 }
```

```
Console
<terminated> B (15) [Java Applica
100
```

Example #2:

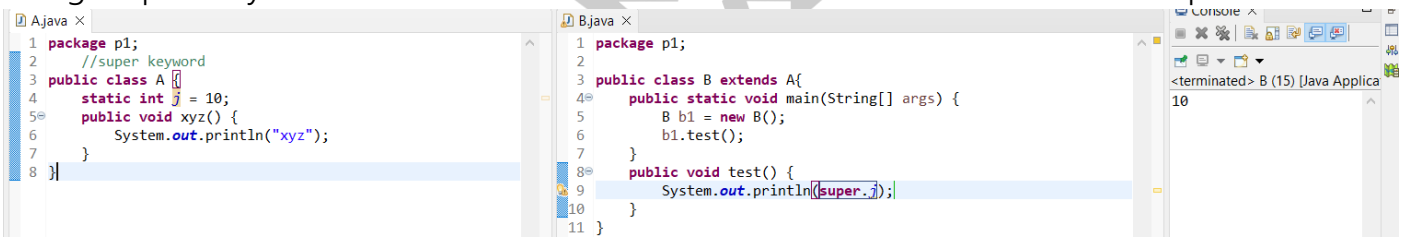


```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     public void xyz() {
5         System.out.println("xyz");
6     }
7 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     public static void main(String[] args) {
5         B b1 = new B();
6         b1.test();
7     }
8     public void test() {
9         super.xyz(); //Accessing parent class method
10    }
11 }
```

```
Console
<terminated> B (15) [Java Applica
xyz
```

Using "super" keyword we can access static and non static members both. Example:

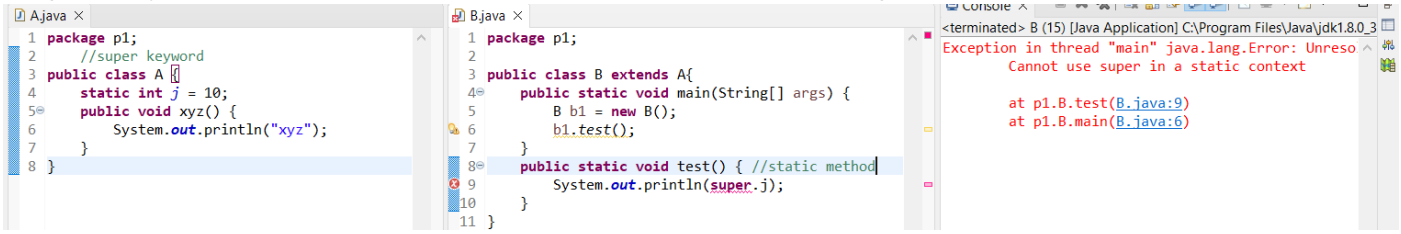


```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     static int j = 10;
5     public void xyz() {
6         System.out.println("xyz");
7     }
8 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     public static void main(String[] args) {
5         B b1 = new B();
6         b1.test();
7     }
8     public void test() {
9         System.out.println(super.j);
10    }
11 }
```

```
Console
<terminated> B (15) [Java Applica
10
```

"super' keyword cannot be used inside static context. Example:



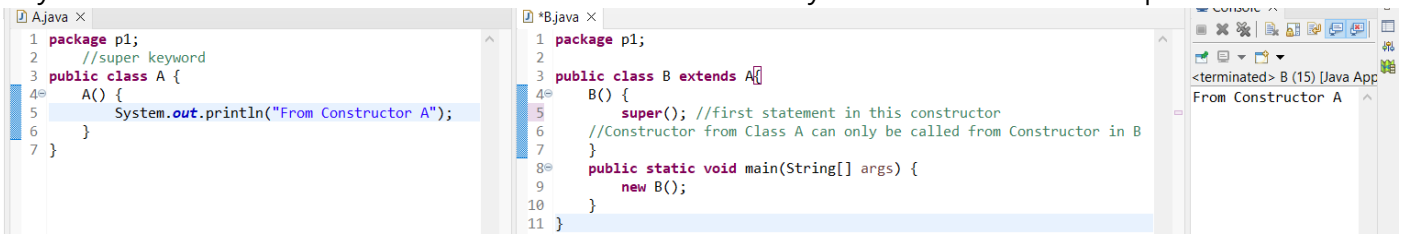
```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     static int j = 10;
5     public void xyz() {
6         System.out.println("xyz");
7     }
8 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     public static void main(String[] args) {
5         B b1 = new B();
6         b1.test();
7     }
8     public static void test() { //static method
9         System.out.println(super.j);
10    }
11 }
```

```
Console
<terminated> B (15) [Java Application] C:\Program Files\Java\jdk1.8.0_3
Exception in thread "main" java.lang.Error: Unreso
        Cannot use super in a static context

        at p1.B.test(B.java:9)
        at p1.B.main(B.java:6)
```

Using "super" keyword, we can call constructor of parent class but then we should use "super" keyword in child class constructor and it should be very first statement. Example:

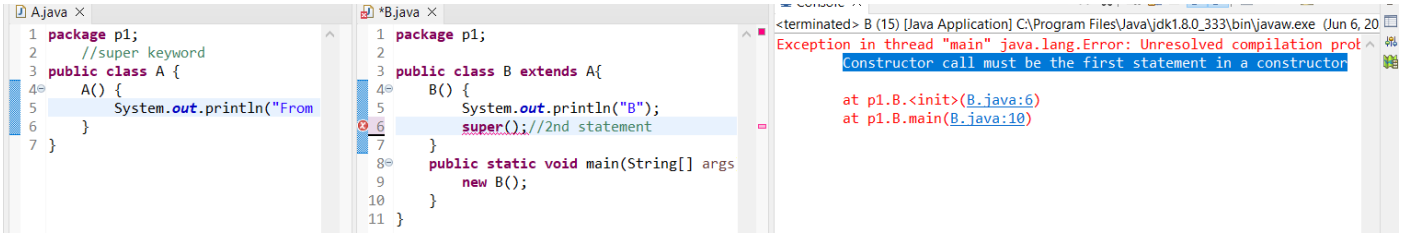


```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() {
5         System.out.println("From Constructor A");
6     }
7 }
```

```
*B.java
1 package p1;
2
3 public class B extends A{
4     B() {
5         super(); //first statement in this constructor
6     //Constructor from Class A can only be called from Constructor in B
7     }
8     public static void main(String[] args) {
9         new B();
10    }
11 }
```

```
Console
<terminated> B (15) [Java App
From Constructor A
```

Constructor call from 2nd statement, Error



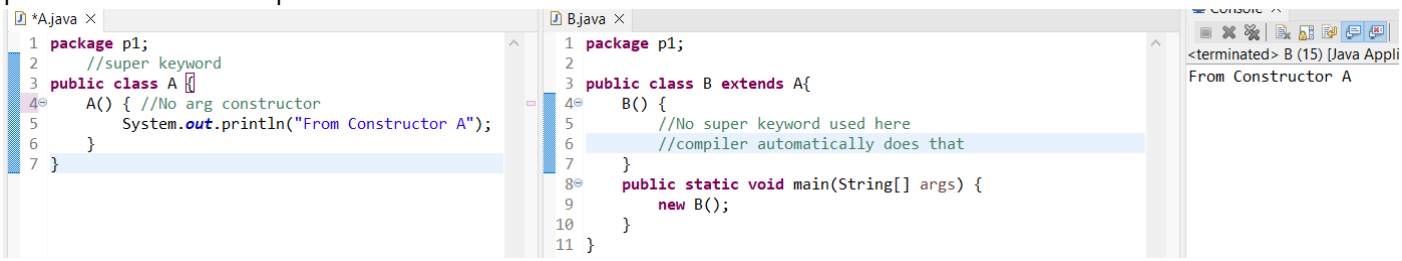
```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() {
5         System.out.println("From
6     }
7 }
```

```
*B.java
1 package p1;
2
3 public class B extends A{
4     B() {
5         System.out.println("B");
6         super();//2nd statement
7     }
8     public static void main(String[] args
9         new B();
10    }
11 }
```

```
<terminated> B (15) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (Jun 6, 20
Exception in thread "main" java.lang.Error: Unresolved compilation prot
        Constructor call must be the first statement in a constructor

        at p1.B.<init>(B.java:6)
        at p1.B.main(B.java:10)
```

If we don't keep "super" keyword inside child class constructor then compiler will automatically place the "super" keyword, such that it can call only no args constructor of parent class. Example:

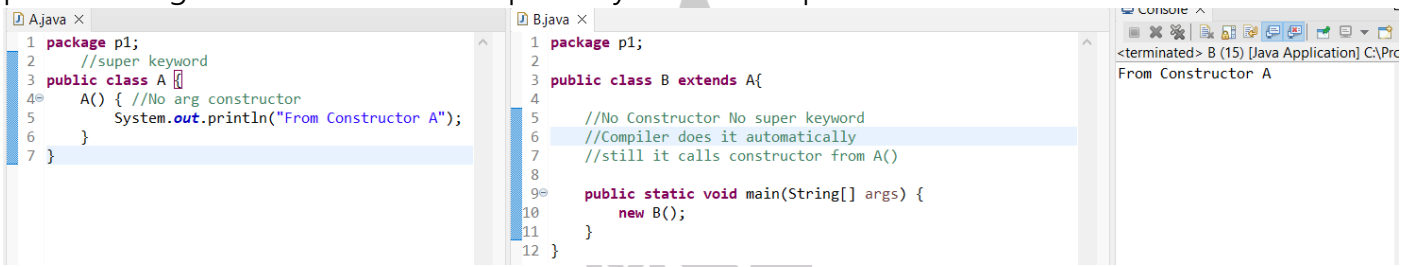


```
*A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() { //No arg constructor
5         System.out.println("From Constructor A");
6     }
7 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     B() {
5         //No super keyword used here
6         //compiler automatically does that
7     }
8     public static void main(String[] args) {
9         new B();
10    }
11 }
```

```
<terminated> B (15) [Java Appli
From Constructor A
```

If we don't create child class constructor with argument, then compiler will automatically place no args constructor with "super" keyword. Example:

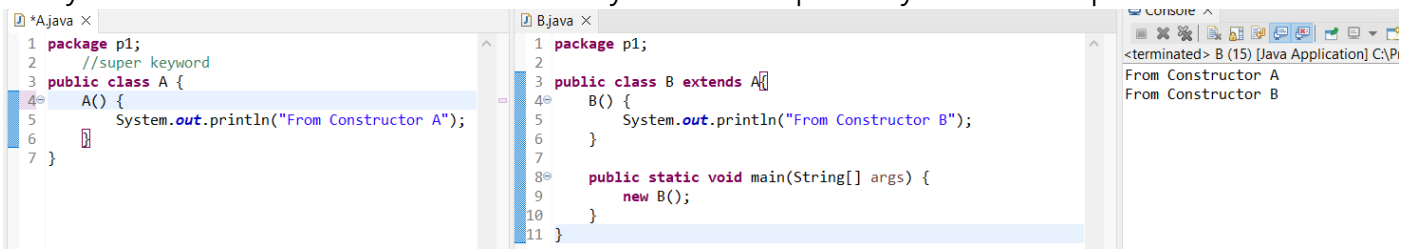


```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() { //No arg constructor
5         System.out.println("From Constructor A");
6     }
7 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4
5     //No Constructor No super keyword
6     //Compiler does it automatically
7     //still it calls constructor from A()
8
9     public static void main(String[] args) {
10        new B();
11    }
12 }
```

```
<terminated> B (15) [Java Application] C:\Pro
From Constructor A
```

Every child class constructor automatically creates "super" keyword. Example:

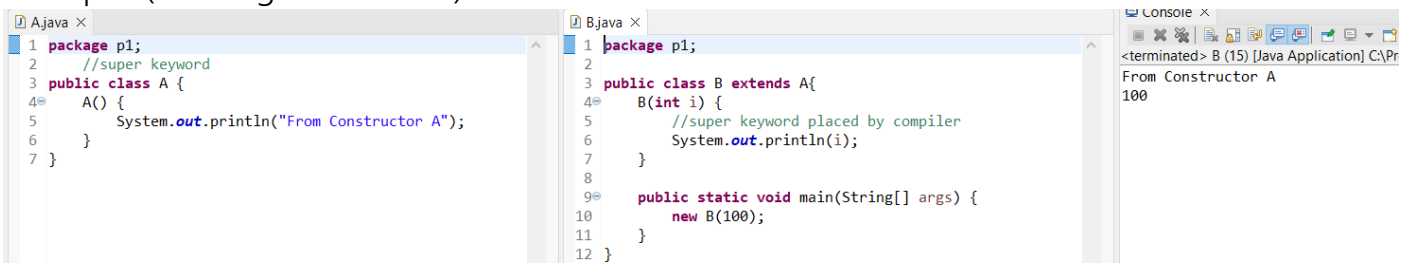


```
*A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() {
5         System.out.println("From Constructor A");
6     }
7 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     B() {
5         System.out.println("From Constructor B");
6     }
7
8     public static void main(String[] args) {
9         new B();
10    }
11 }
```

```
<terminated> B (15) [Java Application] C:\Pr
From Constructor A
From Constructor B
```

Example: (with arg constructor)



```
A.java
1 package p1;
2     //super keyword
3 public class A {
4     A() {
5         System.out.println("From Constructor A");
6     }
7 }
```

```
B.java
1 package p1;
2
3 public class B extends A{
4     B(int i) {
5         //super keyword placed by compiler
6         System.out.println(i);
7     }
8
9     public static void main(String[] args) {
10        new B(100);
11    }
12 }
```

```
Console
<terminated> B (15) [Java Application] C:\Pr
From Constructor A
100
```

If in the parent class there's only constructors with argument then as a programmer we should explicitly write "super" keyword in child class constructor. Example:

A.java
```
1 package p1;
2     //super keyword
3 public class A {
4     A(int i) {
5         System.out.println(i);
6     }
7 }
```

B.java
```
1 package p1;
2
3 public class B extends A{
4     B() {
5         //Error, because we didn't write super keyword.
6     }
7
8     public static void main(String[] args) {
9         new B();
10    }
11 }
```

Console
```
<terminated> B (15) [Java Application] C:\Pro
Exception in thread "main" java.
        Implicit super constructo

        at p1.B.<init>(B.java:4)
        at p1.B.main(B.java:9)
```

A.java
```
1 package p1;
2     //super keyword
3 public class A {
4     A(int i) {
5         System.out.println(i);
6     }
7 }
```

*B.java
```
1 package p1;
2
3 public class B extends A{
4     B() {
5         super(100); //No Error
6     }
7
8     public static void main(String[] args) {
9         new B();
10    }
11 }
```

Console
```
<terminated> B (15) [Java Application] C:\Pr
100
```

"super" keyword is not automatically kept when there are only constructors with arguments in parent class, whereas super keyword will be placed automatically in all the child class constructors when in the parent class there's a constructor with no arguments. Example:

A.java
```
1 package p1;
2     //super keyword
3 public class A {
4     A(int i) {
5         System.out.println(i);
6     }
7
8     A() {
9         System.out.println("From No arg Constructor in A
10    }
11 }
```

B.java
```
1 package p1;
2
3 public class B extends A{
4     B() {
5         //There are 2 constructors in parent class
6         //compiler's default super keyword will call
7         //constructor with no args in parent class
8
9         System.out.println("From Child Class");
10    }
11
12    public static void main(String[] args) {
13        new B();
14    }
15 }
```

Console
```
<terminated> B (15) [Java Application] C:\Progr
From No arg Constructor in A
From Child Class
```

We cannot use "this" keyword and "super" keyword in the same constructor to call another constructor as either of the statements becomes 2nd statement then we'll get an error. Ex:

A.java
```
1 package p1;
2     //super keyword
3 public class A {
4     A(int i) {
5         System.out.println(i);
6     }
7
8     A() {
9         System.out.println("From No arg Constructor in A
10    }
11 }
```

B.java
```
1 package p1;
2
3 public class B extends A{
4     B() {
5         System.out.println("From Child Class");
6     }
7
8     B(int i) {
9         this(); super(); //Error
10        System.out.println(i);
11    }
12
13    public static void main(String[] args) {
14        new B(100);
15    }
16 }
```

Console
```
<terminated> B (15) [Java Application] C:\Progr
Exception in thread "main" java.
        Constructor call must be

        at p1.B.<init>(B.java:9)
        at p1.B.main(B.java:14)
```

If child class constructor consist of "this" keyword then in that constructor "super" keyword will not be automatically placed. Example:

A.java
```
1 package p1;
2     //super keyword
3 public class A {
4     A(int i) {
5         System.out.println(i);
6     }
7
8     A() {
9         System.out.println("From No arg Constructor in A
10    }
11 }
```

*B.java
```
1 package p1;
2
3 public class B extends A{
4     B() { //super keyword automatically placed
5         System.out.println("From Child Class");
6     }
7
8     B(int i) { //super keyword not automatically placed
9         this();
10        System.out.println(i);
11    }
12
13    public static void main(String[] args) {
14        new B(100);
15    }
16 }
```

Console
```
<terminated> B (15) [Java Application] C:\Progr
From No arg Constructor in A
From Child Class
100
```

27/05/2022 (Friday)

OOPs CONCEPT: ABSTRACTION (Contd...)

Abstract class doesn't support multiple inheritance, Ex:

C.java
```
1 package p1;
2
3 abstract public class C extends A,B{ //Error
4
5 }
```

B.java
```
1 package p1;
2 abstract public class B {
3 }
```

A.java
```
1 package p1;
2
3 public abstract class A {
4 }
```

Example: *(Inheritance from Abstract class to Java class)*

B.java
```
1 package p1;
2 public class B extends A{
3     public static void main(String[] args) {
4         B b1 = new B();
5         b1.test(); //calling non static method
6         System.out.println(b1.x); //printing non static member
7         b1.example(); //calling overriden abstract method
8         System.out.println(A.y); //calling static member with ClassName
9     }
10
11    @Override
12    public void example() {
13        System.out.println(300);
14    }
15 }
```

A.java
```
1 package p1;
2 abstract public class A {
3     int x = 100; //non static member
4     static int y = 400; //static member
5     public void test() { //non static method
6         System.out.println(200);
7     }
8     abstract public void example(); //abstract method
9 }
```

Console
```
<terminated> B (11
200
100
300
400
```

Points to remember:

- When Inheriting from an abstract Class @Override & Complete.
- If there's a non static member in abstract class, we can inherit & access that.
- If it is static we can directly access with class name.

Q. Difference between Interface and Abstract Class?

A.

Interface		Abstract Class	
1.	Interfaces are 100% incomplete.	1.	Abstract Classes can be 0-100% incomplete.
2.	Supports multiple Inheritance.	2.	Doesn't support multiple Inheritance.
3.	All variables by default are static and final.	3.	Here we can create static/non static variables

EXCEPTIONS IN JAVA

Q. What is exception in JAVA?

A. Whenever a bad user input is given, it will halt the program execution abruptly. This is called as "exception".

Example:

*A.java

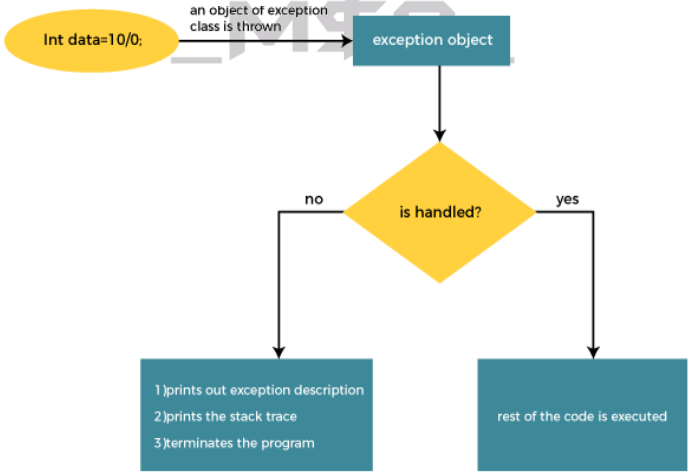
```
1  package p1;
2  public class A {
3      public static void main(String[] args) {
4          int x = 10;
5          int y = 0;
6          int z = x/y; //Exception, Program Execution stops here & did not print next values.
7
8          System.out.println(100);
9          System.out.println(200);
10     }
11 }
```

Console

```
<terminated> A (17) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at p1.A.main(A.java:6)
```

EXCEPTION HANDLING

To handle exception in JAVA, we use Try Catch Block, if an exception occurs in try block, then try block automatically creates an exception object & gives that object's address to catch block. Catch block will now suppress the exception and the further code from there will continue to run.



Example:

A.java

```
1  package p1;
2  public class A {
3      public static void main(String[] args) {
4          try { //type "try" then ctrl+1, & paste code in Try Block
5              int x = 10;
6              int y = 0;
7              int z = x/y; //Exception
8              System.out.println("Welcome"); //Just to check, This line will not run because its in Try Block
9          } catch (Exception e) {
10             System.out.println(e); // it will print reason for exception
11         }
12
13         System.out.println(100);
14         System.out.println(200);
15     }
16 }
```

Console

```
<terminated> A (17) [Java Application] C:\Program Files\Ja
java.lang.ArithmeticException: / by zero
100
200
```

To know the exact line number where exception has occurred, then instead of using "`System.out.println(e);`" we use "`e.printStackTrace();`", Example:

A.java

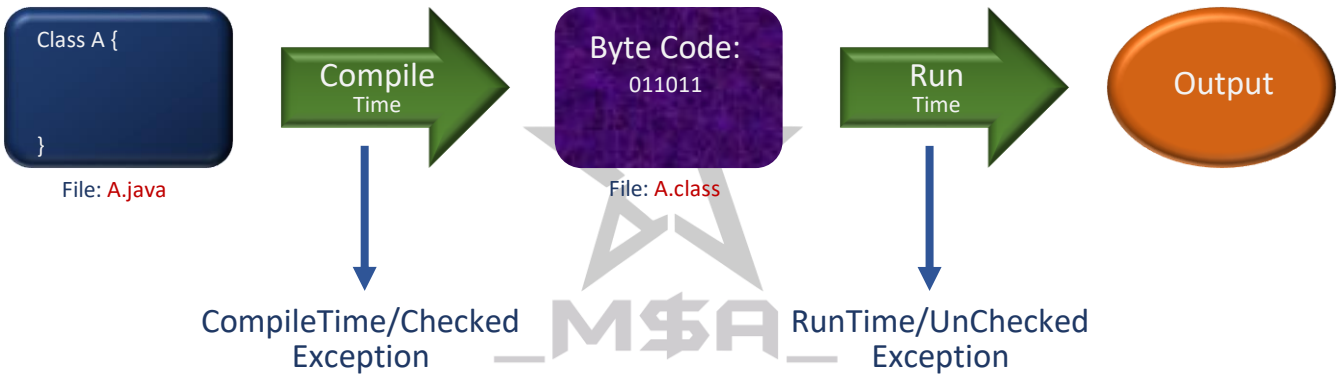
```
1  package p1;
2  public class A {
3      public static void main(String[] args) {
4          try { //type "try" then ctrl+1, & paste code in Try Block
5              int x = 10;
6              int y = 0;
7              int z = x/y; //Exception
8              System.out.println("Welcome"); //Just to check, This line will not run because its in Try Block
9          } catch (Exception e) {
10             e.printStackTrace(); // it will print reason for exception & in which line exception occured.
11         }
12
13         System.out.println(100);
14         System.out.println(200);
15     }
16 }
```

Console

```
<terminated> A (17) [Java Application] C:\Program Files\Ja
java.lang.ArithmeticException: / by zero
        at p1.A.main(A.java:7)
100
200
```

Types of Exceptions

There are mainly 2 types of exceptions, RUNTIME EXCEPTION and COMPILE TIME EXCEPTION



CompileTime/Checked Exception: These exceptions will occur at compile time (when ".JAVA" file is compiled to ".CLASS" file). These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

For example, if we use FileReader class in the program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception, see the below example:

*A.java

```
1  package p1;
2  import java.io.FileReader;
3
4  public class A {
5      public static void main(String[] args) {
6          FileReader fr = new FileReader("D://test.txt");
7          //FileNotFoundException
8      }
9  }
10
```

Unhandled exception type FileNotFoundException

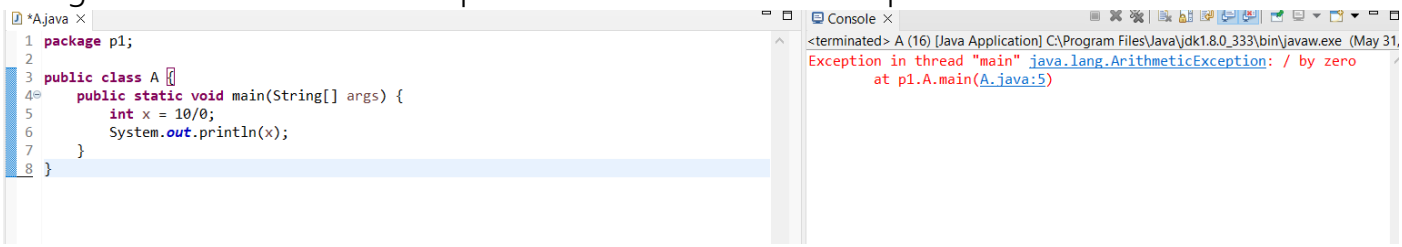
2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

RunTime/UnChecked Exception: These exceptions occurs when we run ".CLASS" file. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. See the below example:



Q. Difference between CompileTime and RunTime Errors?

A.

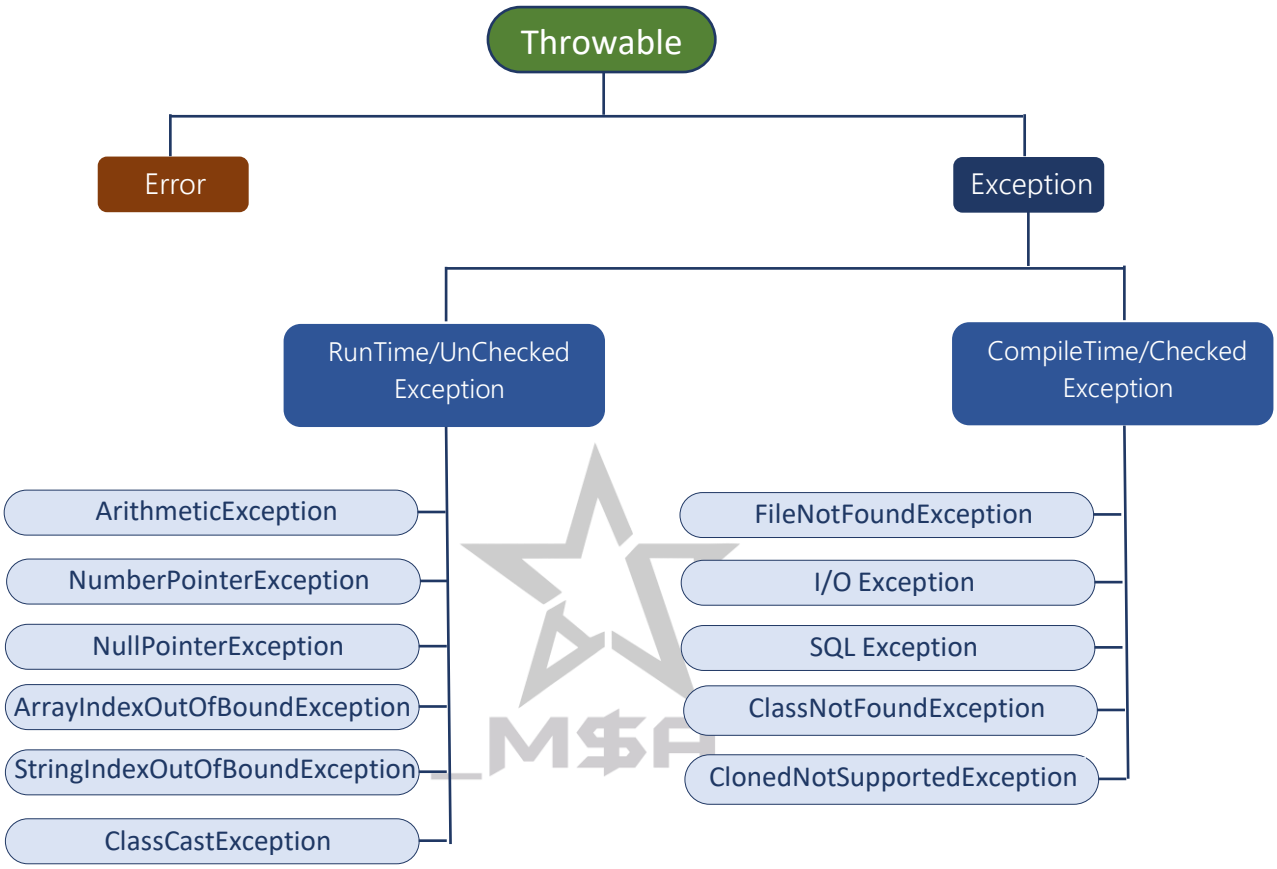
CompileTime/Checked		RunTime/UnChecked	
1.	The compile-time errors are the errors which are produced at the compile-time, and they are detected by the compiler.	1.	The runtime errors are the errors which are not generated by the compiler and produce an unpredictable result at the execution time.
2.	In this case, the compiler prevents the code from execution if it detects an error in the program.	2.	In this case, the compiler does not detect the error, so it cannot prevent the code from the execution.
3.	It contains the syntax and semantic errors such as missing semicolon at the end of the statement.	3.	It contains the errors such as division by zero, determining the square root of a negative number.

30/05/2022 (Monday)

EXCEPTION CLASS HIERARCHY IN JAVA

Q. What are Exception Class Hierarchy in JAVA?

A.



31/05/2022 (Tuesday)

EXCEPTION CLASS HIERARCHY IN JAVA (Contd…)
RunTime Exception

ArithmeticException: This class can handle ArithmeticExcpetion like dividing a number by zero.

Ex:

A.java

```
1  package p1;
2      //ArithmeticException
3  public class A {
4      public static void main(String[] args) {
5          try {
6              int x = 10;
7              int y = 0;
8              int z = x/y;
9              System.out.println(z);
10         } catch (ArithmeticException e) {
11             e.printStackTrace();
12         }
13         System.out.println("Welcome");
14     }
15 }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Ja
java.lang.ArithmeticException: / by zero
        at p1.A.main(A.java:8)
Welcome
```

NumberFormatException: When an invalid String to number conversion is done we get NumberFormatException as shown in below example:

A.java

```
1  package p1;
2      //NumberFormatException
3  public class A {
4      public static void main(String[] args) {
5          try {
6              String x = "100xyz"; //only numeric value is allowed here.
7              int a = Integer.parseInt(x); //For Conversion, using Wrapper Class
8          } catch (NumberFormatException e) {
9              e.printStackTrace();
10         }
11         System.out.println("Welcome");
12     }
13 }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 31, 2022, 11:03:5
java.lang.NumberFormatException: For input string: "100xyz"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at p1.A.main(A.java:7)
Welcome
```

A.java

```
1  package p1;
2      //No NumberFormatException, No Error, No Try Block
3  public class A {
4      public static void main(String[] args) {
5              String x = "100"; //only numeric value is allowed here.
6              int a = Integer.parseInt(x); //For Conversion, using Wrapper Class
7              System.out.println(a);
8              System.out.println("Welcome");
9      }
10 }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe  (May 31, 2022, 11:28:0
100
Welcome
```

Example: *Conversion String to Float*

A.java

```
1  package p1;
2      //Conversion String to float
3  public class A {
4      public static void main(String[] args) {
5              String x = "10.3"; //only float value is allowed here.
6              float a = Float.parseFloat(x); //For Conversion, using Wrapper Class
7              System.out.println(a);
8              System.out.println("Welcome");
9      }
10 }
```

Console

```
<terminated> A (16) [Java Application]
10.3
Welcome
```

Example: *Conversion String to boolean*

A.java

```
1  package p1;
2      //Conversion String to boolean
3  public class A {
4      public static void main(String[] args) {
5              String x = "true"; //any value other than "true" will result false
6              boolean a = Boolean.parseBoolean(x); //For Conversion, using Wrapper Class
7              System.out.println(a);
8      }
9  }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 31, 2(
true
```

A.java

```
1  package p1;
2      //Conversion String to boolean
3  public class A {
4      public static void main(String[] args) {
5              String x = "123xyz"; //any value other than "true" will result false
6              boolean a = Boolean.parseBoolean(x); //For Conversion, using Wrapper Class
7              System.out.println(a);
8      }
9  }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 31, 2(
false
```

NullPointerException: When we access non static member with null reference variable we get "NullPointerException" as shown in the below example:

A.java

```
1  package p1;
2      //NullPointerException
3  public class A {
4      int x = 10;
5      public static void main(String[] args) {
6          A a1 = null;
7          System.out.println(a1.x);
8      }
9  }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 31, 2(
Exception in thread "main" java.lang.NullPointerException
        at p1.A.main(A.java:7)
```

A.java

```
1  package p1;
2      //NullPointerException Exception Handling
3  public class A {
4      int x = 10;
5      public static void main(String[] args) {
6          try {
7              A a1 = null;
8              System.out.println(a1.x);
9          } catch (NullPointerException e) {
10             e.printStackTrace();
11         }
12         System.out.println("Welcome");
13     }
14 }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 31, 2(
java.lang.NullPointerException
        at p1.A.main(A.java:8)
Welcome
```

We can also get result with just writing "Exception e" in catch block, Example:

A.java

```
1  package p1;
2      //NullPointerException Exception Handling
3  public class A {
4      int x = 10;
5      public static void main(String[] args) {
6          try {
7              A a1 = null;
8              System.out.println(a1.x);
9          } catch (Exception e) {
10             e.printStackTrace();
11         }
12         System.out.println("Welcome");
13     }
14 }
```

Console

```
<terminated> A (16) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (May 31, 2(
java.lang.NullPointerException
        at p1.A.main(A.java:8)
Welcome
```

Q. What's the difference between using "Exception" class & "NumberFormatException" class in catch block?

A. When we use NumberFormatException class in catch block, then it can handle only NumberFormatException, but if we use Exception class in catch block, then it can handle all the Exceptions that occur in try block.

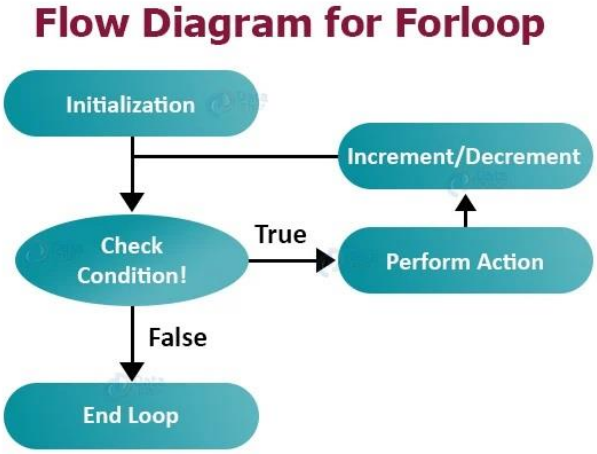
LOOPS IN JAVA

Loops are used to repeat a block of code, for example, if we want to show a message 100 times, then rather than typing the same code 100 times, we can use a loop.

There are mainly 3 types of Loops:

1. For loop
2. While loop
3. Do-While loop

For Loop: In JAVA for loop is used to run a block of code for a certain number of times.



Example #1:

A.java

```
1  package p1;
2      //Loop: For Loop
3  public class A {
4      public static void main(String[] args) {
5          for(int i=0;i<10;i++) {
6              //i=0 is initial vaule, Initialization
7              //i<5 is Condition
8              //i++ is increment, updating the initial value by 1
9              //so it prints the value of i until i<10 only, when i=10, it stops
10             System.out.println(i);
11         }
12         System.out.println("For Loop Ended here");
13     }
14 }
```

Console

```
<terminated> A (16) [Java Application] C:\P
0
1
2
3
4
5
6
7
8
9
For Loop Ended here
```

Example #2:

A.java

```
1  package p1;
2      //Loop: For Loop
3  public class A {
4      public static void main(String[] args) {
5          for(int i=10;i>=0;i--) {
6              //i=10 is initial vaule, Initialization
7              //i>=0 is Condition (Less than or Equal to Zero)
8              //i-- is decrement, decreasing the initial value by 1
9              //so it prints the value of i until i>=0 only, when i=0, it stops
10             System.out.println(i);
11         }
12         System.out.println("For Loop Ended here");
13     }
14 }
```

Console

```
<terminated> A (16) [Java Application] C:\P
10
9
8
7
6
5
4
3
2
1
0
For Loop Ended here
```

Example #3:

A.java

```
1  package p1;
2      //Loop: For Loop
3  public class A {
4      public static void main(String[] args) {
5          for(int i=1;i<10;i++) {
6              //i=1 is initial vaule, Initialization
7              //i<=10 is Condition (Less than 10)
8              //i++ is Iincrement, increasing the initial value by 1
9              //so it repeats whatever the code is written in "for block" until i<=10 only, when i=10, it stops.
10             System.out.println("Pankaj Sir Academy");
11         }
12         System.out.println(""); //just to skip one line in output
13         System.out.println("For Loop Ended here");
14     }
15 }
```

Console

```
<terminated> A (16) [Java Application] C:\P
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy
Pankaj Sir Academy

For Loop Ended here
```

"break" Keyword

"break" Keyword is used to terminate the "loops" and "switch statements" in java. When the break keyword is used within a loop, the loop is immediately terminated and the program control goes to the next line of codes after the loop. Example:

A.java

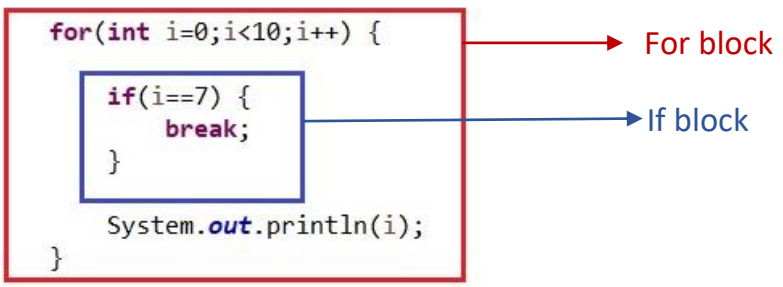
```
1  package p1;
2      //Break
3  public class A {
4      public static void main(String[] args) {
5          for(int i=0;i<10;i++) {
6              //i=1 is initial vaule, Initialization
7              //i<=10 is Condition (Less than 10)
8              //i++ is Iincrement, increasing the initial value by 1
9              //so it prints the value of i until i<=10 only, when i=10, it should stop.
10             //But here we are going to use 'break".
11             if(i==7) { //"==" used to compare the value, "=" to store the value
12                 break; //so here when i's value reaches 7 (which is true), the loop will stop
13             }
14             System.out.println(i);
15         }
16         System.out.println(""); //just to skip one line in output
17         System.out.println("For Loop Ended here");
18     }
19 }
```

Console

```
<terminated> A (16) [Java Appli
0
1
2
3
4
5
6

For Loop Ended here
```

Labelled Break: Break keyword will always exit "for block", but with labelled break we can specify which block to exit in the loop, "for block" or "if block".



Example #1: *(Exiting "if block")*

A.java B.java

```
1  package p1;
2      //Labelled Break
3  public class B {
4      public static void main(String[] args) {
5
6          for(int i=0;i<10;i++) {
7
8              z: if(i==5) { //applying lablelled break on 'if block', when i=5, break exit if block
9                  break z;
10             }
11
12             System.out.println(i);
13         }
14         System.out.println(""); //just to skip one line in output
15         System.out.println("Loop Ended here");
16     }
17 }
```

Console

```
<terminated> B (11) [Java Applic
0
1
2
3
4
5
6
7
8
9

Loop Ended here
```

Example #2: *(Exiting "for block")*

A.java B.java

```
1  package p1;
2      //Labelled Break
3  public class B {
4      public static void main(String[] args) {
5
6          z: for(int i=0;i<10;i++) { //applying lablelled break on 'for block', when i=5, break exits "for block".
7
8              if(i==5) {
9                  break z;
10             }
11
12             System.out.println(i);
13         }
14         System.out.println(""); //just to skip one line in output
15         System.out.println("Loop Ended here");
16     }
17 }
```

Console

```
<terminated> B (11) [Java Applica
0
1
2
3
4

Loop Ended here
```

01/06/2022 (Wednesday)

"Continue" Keyword

The "continue" keyword is used to end the current iteration (repetition) in a for loop (or a while loop), and continues to the next iteration (repetition). Example:

A.java | B.java

```
1  package p1;
2      //Continue Keyword
3  public class B {
4      public static void main(String[] args) {
5          for (int i = 0; i <10; i++) {
6              if(i==6) {
7                  continue; //It ends current loop & goes back to for (Line#5), thus it skips the next value i=6.
8              }
9              System.out.println(i);
10         }
11         System.out.println("Continue Keyword didn't print 6");
12     }
13 }
```

Console

```
<terminated> B (11) [Java Application] C:\Progra
0
1
2
3
4
5
7
8
9
Continue Keyword didn't print 6
```

CONDITIONAL STATEMENTS IN JAVA

If and Else Statements: The Java if statement tests the condition. It executes the "if block" if condition is true, otherwise "else" block is executed. Example:

A.java | B.java

```
1  package p1;
2      //If-Else Statement
3  public class B {
4      public static void main(String[] args) {
5          int x= 10;
6          //Example #1
7          if(x>5) { //True
8              System.out.println("YES, Greater than 5");
9          }else {
10             System.out.println("NO, Less than 5");
11         }
12         //Example #2
13         if(x<5) { //False
14             System.out.println("Wrong");
15         }else {
16             System.out.println("NO, Not less than 5");
17         }
18     }
19 }
```

Console

```
<terminated> B (11) [Java Application] C:\
YES, Greater than 5
NO, Not less than 5
```

Else-if Statement: The Java if statement tests the "if block" condition, if it is false the control goes to "Else-If block", If it is true it prints it's value, if not it prints "else block" value.
Note: we can have multiple Else-if statements. Example:

A.java | *B.java

```
1  package p1;
2      //Else If Statement
3  public class B {
4      public static void main(String[] args) {
5          int x= 100;
6
7          if(x==99) {
8              System.out.println("A");
9          }else if(x==100) { //True
10             System.out.println("B");
11         }else if(x==200) {
12             System.out.println("C");
13         }else {
14             System.out.println("D");
15         }
16     }
17 }
```

Console

```
<terminated> B (11) [Java Application] C:\P
B
```

Switch Statement: The switch statement tests the equality of a variable against multiple values.

Points to Remember:

- There can be one or any number of case values for a switch expression.
- Each case statement can have a break statement which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a default label which is optional.

Example #1:

A.java B.java

```
package p1;
    //Switch Statement
public class B {
    public static void main(String[] args) {
        int key = 3; //This value is present in Case3, so it'll directly print Case3's value
        switch (key) {
        case 1:
            System.out.println(1);
            break;
        case 2:
            System.out.println(2);
            break;
        case 3:
            System.out.println(3);
            break;
        case 4:
            System.out.println(4);
            break;
        default:
            System.out.println("Value not found");
            break;
        }

    }
}
```

Console

```
<terminated> B (11) [Java Application] C:\Program Files
3
```

Example #2: *(If No value found in switch statements)*

A.java B.java

```
package p1;
    //Switch Statement
public class B {
    public static void main(String[] args) {
        int key = 10; //This value is not present in any Case, so it'll directly print Default's value
        switch (key) {
        case 1:
            System.out.println(1);
            break;
        case 2:
            System.out.println(2);
            break;
        case 3:
            System.out.println(3);
            break;
        case 4:
            System.out.println(4);
            break;
        default:
            System.out.println("Value not found");
            break;
        }

    }
}
```

Console

```
<terminated> B (11) [Java Application] C:\Program Files
Value not found
```

Example #3: *(If No break given)*

A.java B.java

```
package p1;
    //Switch Statement
public class B {
    public static void main(String[] args) {
        int key = 3; //This value is present in Case3, so it'll directly print Case3's value
        switch (key) {
        case 1:
            System.out.println(1);
            break;
        case 2:
            System.out.println(2);
            break;
        case 3:
            System.out.println(3);
                //No break in Case 3, so it will print 4 also
        case 4:
            System.out.println(4);
            break;
        default:
            System.out.println("Value not found");
            break;
        }

    }
}
```

Console

```
<terminated> B (11) [Java Application] C:\Program Files
3
4
```

LOOPS IN JAVA (Contd...)

While Loop: The while loop is used to repeat a section of code n number of times until a specific condition is met.

```
1  package p1;
2      //While Loop
3  public class B {
4      public static void main(String[] args) {
5          int x = 0;
6          while (x<3) { //it prints unless the x value is less than 3
7              System.out.println(x);
8              x++; //If we forget this, it will keep on printing 0.
9          }
10     }
11 }
```

```
<terminated> B (11) [Java Application] C:\
0
1
2
```

Example #2: *(If the condition is false, it prints nothing)*

```
1  package p1;
2      //While Loop
3  public class B {
4      public static void main(String[] args) {
5          int x = 5;
6          while (x<3) { //condition: false, program stops here & prints nothing
7              System.out.println(x);
8          }
9      }
10 }
```

```
<terminated> B (11) [Java Application] C:\
```

Do While Loop: Java do-while loop is called an exit control loop. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. Example:

```
1  package p1;
2      //Do While Loop
3  public class B {
4      public static void main(String[] args) {
5          int x = 10;
6          do {    //here it won't check the condition
7              System.out.println(x);
8          } while (x<5); //here it checks the condition, if true it runs, if false it won't run
9      }
10 }
```

```
<terminated> B (11) [Java Application] C:\P
10
```

Example #2:

```
1  package p1;
2      //Do While Loop
3  public class B {
4      public static void main(String[] args) {
5          int x = 0;
6          do {    //here it won't check the condition
7              System.out.println(x);
8              x++; //if we forget this step, it will keep on printing "0".
9          } while (x<5); //true: enters the loop
10     }
11 }
```

```
<terminated> B (11) [Java Application] C:\Pr
0
1
2
3
4
```

USER INPUT (SCANNER CLASS) IN JAVA

The Scanner class is used to get user input, and it is found in the java.util package. To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings. Example:

D.java

```
1  package p1;
2      //Scanner Class (User Input)
3  import java.util.Scanner; //Default Package available for user inputs
4
5  public class D {
6      public static void main(String[] args) {
7          Scanner s = new Scanner(System.in); //System.out for printing, System.in for reading input, this is default syntax.
8          System.out.println("Enter Your Name");
9          String name = s.next(); //type "s.next()" then Ctrl+1 to assign local variable, here s=variable name.
10         System.out.println("Your name is: " + name); //To print return text
11         //this method is called "Concatenation", It joins two string values & return it
12     }
13 }
```

Console

```
<terminated> D (3) [Java Application] C:\P
Enter Your Name
Mike
Your name is: Mike
```

The previous program cannot take two words, If your name is “Mike Tyson” it can only print Mike, the next example shows how to take multiple word string values.

Example #2:

D.java

```
1  package p1;
2      //Scanner Class (User Input)
3  import java.util.Scanner; //Default Package available for user inputs
4
5  public class D {
6      public static void main(String[] args) {
7          Scanner s = new Scanner(System.in);
8          System.out.println("Enter Your Name");
9          String name = s.nextLine(); //nextLine() used to store multiple word string
10         System.out.println("Your name is: " + name);
11     }
12 }
```

Console

```
<terminated> D (3) [Java Application] C:\Pr
Enter Your Name
Mike Tyson
Your name is: Mike Tyson
```

Example #3:

D.java

```
1  package p1;
2      //Scanner Class (User Input)
3  import java.util.Scanner; //Default Package available for user inputs
4
5  public class D {
6      public static void main(String[] args) {
7          //Name
8          Scanner s = new Scanner(System.in);
9          System.out.println("Enter Your Name");
10         String name = s.nextLine(); //nextLine() used to store multiple word string
11         System.out.println("Your name is: " + name);
12
13         //Age
14         System.out.println("Enter Your Age");
15         int age = s.nextInt(); //nextInt() used to store integer value
16         System.out.println("Your Age is: " + age);
17
18         //Weight
19         System.out.println("Enter Your Weight");
20         float weight = s.nextFloat(); //nextFloat() used to store float value
21         System.out.println("Your Weight is: " + weight);
22
23         //True/False
24         System.out.println("The Sky is Blue");
25         boolean ans = s.nextBoolean(); //nextFloat() used to store float value
26         System.out.println("Your Answer is: " + ans);
27     }
28 }
```

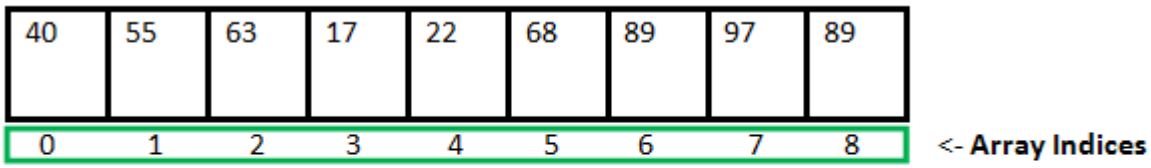
Console

```
<terminated> D (3) [Java Application] C:\P
Enter Your Name
Mike Tyson
Your name is: Mike Tyson
Enter Your Age
25
Your Age is: 25
Enter Your Weight
78.25
Your Weight is: 78.25
The Sky is Blue
True
Your Answer is: true
```

02/06/2022 (Thursday)

ARRAY IN JAVA

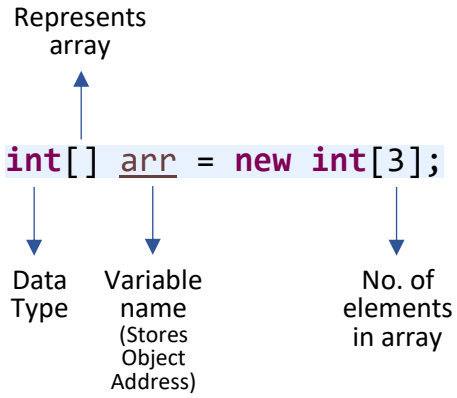
Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

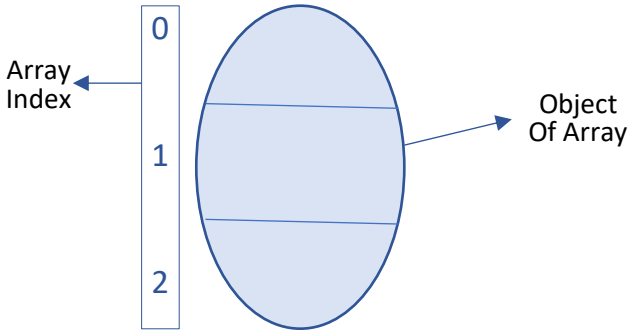


Array Length = 9
First Index = 0
Last Index = 8

A Java array variable can also be declared like other variables with [] after the data type.
Advantages:

- Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.





A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[3];
6          arr[0] = 100;
7          arr[1] = 200;
8          arr[2] = 300;
9
10         System.out.println(arr[2]);
11         System.out.println(arr[0]);
12         System.out.println(arr[1]);
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Prograr
300
100
200
```

EXCEPTION: RunTime/UnChecked: ArrayIndexOutOfBoundsException

Ex: There's only 3 Indexes (0,1,2) if we print 3, it gives "ArrayIndexOutOfBoundsException"

*A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[3];
6          arr[0] = 100;
7          arr[1] = 200;
8          arr[2] = 300;
9
10         System.out.println(arr[2]);
11         System.out.println(arr[0]);
12         System.out.println(arr[1]);
13         System.out.println(arr[3]);
14         //There's only 3 indices (0,1,2)
15         //if we print 3, it gives "ArrayIndexOutOfBoundsException"
16     }
17 }
```

Console

```
<terminated> A (19) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (Jun 3, 2022
300
100
200
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at p1.A.main(A.java:13)
```

Example #3: Let's suppose there are 100s of elements in an array, instead of writing "`System.out.println(arr[#]);`" 100 times we can use here "Loop". Example:

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[3];
6          arr[0] = 100;
7          arr[1] = 200;
8          arr[2] = 300;
9
10     for (int i=0; i<3; i++) {
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
100
200
300
```

Example #4: Instead of writing "`i<3;`" in loop condition we can write "`i<arr.length;`" it will automatically take the length of array.

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[3];
6          arr[0] = 100;
7          arr[1] = 200;
8          arr[2] = 300;
9
10         for (int i=0; i<arr.length; i++) { //here "arr.length=3"
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pro
100
200
300
```

Example #5: in array If we don't initialize the value, depending upon the data type it will print default value, here int default value is "0".

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[5];
6          arr[0] = 100;
7          arr[1] = 200;
8          arr[2] = 300;
9
10         for (int i=0; i<arr.length; i++) {
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pro
100
200
300
0
0
```

Example #5: *(String Value, Default String Value = null)*

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          String[] arr = new String[5];
6          arr[0] = "Mike Tyson";
7          arr[1] = "My Name";
8          arr[2] = "Your Name";
9
10         for (int i=0; i<arr.length; i++) {
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
Mike Tyson
My Name
Your Name
null
null
```

Example #6: *(Float value, Default Float value = 0.0)*

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          float[] arr = new float[5];
6          arr[0] = 10.3f;
7          arr[1] = 20.5f;
8          arr[2] = 100.25f;
9
10         for (int i=0; i<arr.length; i++) {
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
10.3
20.5
100.25
0.0
0.0
```

Example #7: *(char value, Default value value = Empty Space)*

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          char[] arr = new char[5];
6          arr[0] = 'a';
7          arr[1] = 'b';
8          arr[2] = 'c';
9
10         for (int i=0; i<arr.length; i++) {
11         System.out.println(arr[i]);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pro
a
b
c
```

Example #8: *(foreach loop prints array's each value with just a single statement, it will run automatically depending upon the size of an array's elements), Advantage: We are not worried about how many time loops will repeat.*

```
for
}
```

- for - iterate over collection
- for - use index on array
- for - use index on array with temporary variable
- foreach - iterate over an array or Iterable
- formap - iterate over map
- for
- ForegroundAction - javax.swing.text.StyledEditorKit
- ForkJoinPool - java.util.concurrent
- ForkJoinTask - java.util.concurrent
- ForkJoinWorkerThread - java.util.concurrent
- ForkJoinWorkerThreadFactory - java.util.concurrent.ForkJoinPool
- Form - iava.text.Normalizer

Press 'Ctrl+Space' to show Template Proposals

```
for (char c : arr) {

}
```

Press 'Tab' from proposal table or click for focus

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[5];
6          arr[0] = 10;
7          arr[1] = 20;
8          arr[2] = 30;
9
10         for (int c : arr) { //here arr = lenght of array & all values of arrays stored in it
11             System.out.println(c);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
10
20
30
0
0
```

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = new int[5];
6          arr[0] = 10;
7          arr[1] = 20;
8          arr[2] = 30;
9
10         for (int c : arr) { //here arr = lenght of array & all values of arrays stored in it
11             System.out.println("Hello World");
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
Hello World
Hello World
Hello World
Hello World
Hello World
```

Example #9: *(With String)*

A.java

```
1  package p1;
2      //Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          String[] arr = new String[3];
6          arr[0] = "Mike";
7          arr[1] = "Tyson";
8          arr[2] = "John";
9
10         for (String c : arr) {
11             System.out.println(c);
12         }
13     }
14 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
Mike
Tyson
John
```

DYNAMIC ARRAY IN JAVA

The dynamic array is a variable size list data structure. It grows automatically when we try to insert an element if there is no more space left for the new element. It allows us to add and remove elements.

A.java

```
1  package p1;
2      //Dynamic Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          int[] arr = {100, 200, 300}; //It grows dynamically
6
7          for (int c : arr) {
8              System.out.println(c);
9          }
10     }
11 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
100
200
300
```

A.java

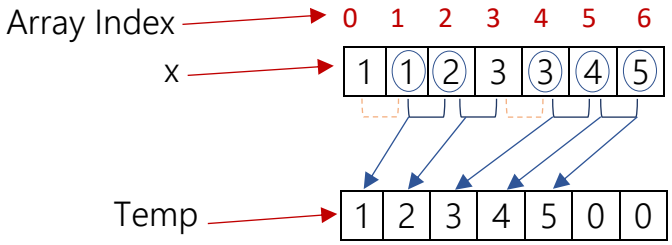
```
1  package p1;
2      //Dynamic Array in JAVA
3  public class A {
4      public static void main(String[] args) {
5          char[] arr = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'}; //It grows dynamically
6
7          for (char c : arr) {
8              System.out.println(c);
9          }
10     }
11 }
```

Console

```
<terminated> A (19) [Java Application] C:\Pr
a
b
c
d
e
f
g
h
i
j
```

03/06/2022 (Friday)

REMOVING DUPLICATE ELEMENTS



In the above example there's a set of elements which is already sorted in ascending order, we're removing duplicate elements from that array.

Let's suppose x=given array which have 7 elements (array index: 0 to 6), first create same size of a temp array, comapre x array index 0 & 1 if the elements are different copy the element in temp array here the element values are same so we're not copying 1, then compare 1 & 2 here the the elements are different so we're copying 1 into temp array, next compare 2 & 3, value is different copy that into temp array, likewise all the elements should be done, & the left over indexes in temp array should be filled with zeros.

See the below examples:

Example #1: *(with error: ArrayIndexOutOfBoundsException):*

A.java

```
1  package p1;
2      //Removing duplicates from an array
3  public class A {
4      public static void main(String[] args) {
5          int[] x = {1,1,2,3,3,4,5};
6          int[] temp = new int[x.length]; //temporary array
7
8          for (int i = 0; i < x.length; i++) { //here x.length = 7
9              //loop will run till 6<7, but when i=6 the i++=7, Error
10             if(x[i]!=x[i+1]) { //comparing the values, "!=" indicates "not equal to"
11
12             }
13         }
14     }
15 }
```

Console

```
<terminated> A (20) [Java Application] C:\Program Files\Java\jdk1.8.0_333\bin\javaw.exe (Jun 4, 20
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
        at p1.A.main(A.java:10)
```

Let's correct it in below example & further copying the elemnt into temp array.

Example #2:

A.java

```
1  package p1;
2      //Removing duplicates from an array
3  public class A {
4      public static void main(String[] args) {
5          int[] x = {1,1,2,3,3,4,5};
6          int[] temp = new int[x.length]; //temporary array
7          int j =0; //to compare & copy the elements from x array
8          for (int i = 0; i < x.length-1; i++) { //now here x.length = 7-1 = 6
9              //loop will run till 5<6, & when i=5 the i++=6, No Error
10             if(x[i]!=x[i+1]) { //comparing the values, "!=" indicates "not equal to"
11                 temp[j]= x[i];
12                 j++;
13             }
14         }
15         for (int b : temp) { //copying the value of temp to print
16             System.out.println(b);
17         }
18     }
19 }
```

Console

```
<terminated> A (20) [Java Application] C:\P
1
2
3
4
0
0
0
```

The last value of x array (which is 5) is not copied because there were no further elements to compare with, so let's copy the last element as it is into temp.

Example #3:

A.java

```
1  package p1;
2      //Removing duplicates from an array
3  public class A {
4      public static void main(String[] args) {
5          int[] x = {1,1,2,3,3,4,5};
6          int[] temp = new int[x.length]; //temporary array
7          int j =0; //to compare & copy the elements from x array
8          for (int i = 0; i < x.length-1; i++) { //now here x.length = 7-1 = 6
9              //loop will run till 5<6, & when i=5 the i++=6, No Error
10             if(x[i]!=x[i+1]) { //comparing the values, "!=" indicates "not equal to"
11                 temp[j]= x[i];
12                 j++;
13             }
14         }
15         temp[j] = x[x.length-1]; //to copy last index as it is, "length-1" is always last index
16         for (int i : temp) { //copying the value of temp to print
17             System.out.println(i);
18         }
19     }
20 }
```

Console

```
<terminated> A (20) [Java Application] C:\Pro
1
2
3
4
5
0
0
```

But as we know there were two duplicate elements so after removing them we got two default int values in place of those elements, so let's remove the zeros.

A.java

```
1  package p1;
2      //Removing duplicates from an array
3  public class A {
4      public static void main(String[] args) {
5          int[] x = {1,1,2,3,3,4,5};
6          int[] temp = new int[x.length]; //temporary array
7          int j =0; //to compare & copy the elements from x array
8          for (int i = 0; i < x.length-1; i++) { //now here x.length = 7-1 = 6
9              //loop will run till 5<6, & when i=5 the i++=6, No Error
10             if(x[i]!=x[i+1]) { //comparing the values, "!=" indicates "not equal to"
11                 temp[j]= x[i];
12                 j++;
13             }
14         }
15         temp[j] = x[x.length-1]; //to copy last index as it is, "length-1" is always last index
16         for (int z = 0; z <j+1; z++) { //to removes zeros, as j's value was 4 so adding +1
17             System.out.println(temp[z]);
18         }
19     }
20 }
```

Console

```
<terminated> A (20) [Java Application] C:\Pro
1
2
3
4
5
```

SWAPPING IN JAVA

Swapping two variables refers to mutually exchanging the values of the variables. Generally, this is done with the data in memory. The simplest method to swap two variables is to use a third temporary variable. Example:

*B.java

```
1  package p1;
2      //Swapping in JAVA
3  public class B {
4      public static void main(String[] args) {
5          int x = 10;
6          int y = 20;
7
8          int temp = x; //temporarily storing the value of x here, otherwise it'll be lost in next step
9
10         x=y; //here "x" has gained the value of x
11         y=temp; //here in "y" we restored value of "x" which was stored in "temp".
12
13         System.out.println(x);
14         System.out.println(y);
15     }
16 }
```

Console

```
<terminated> B (14) [Java Application] C:\Pro
20
10
```

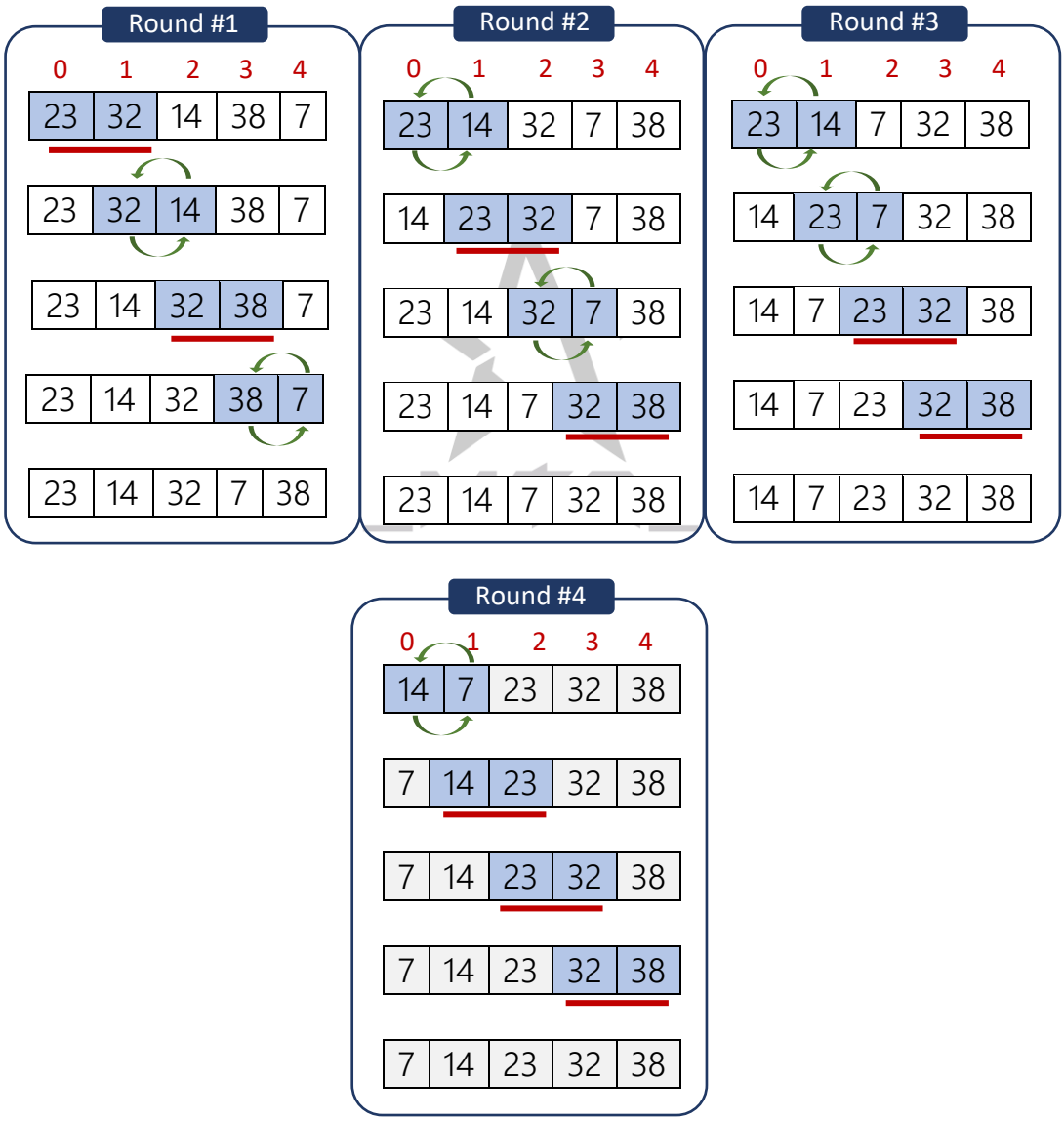
04/06/2022 (Saturday)

SORTING AN ARRAY IN JAVA

The sorting is a way to arrange elements of a list or array in a certain order. The order may be in ascending or descending order. The numerical and lexicographical (alphabetical) order is a widely used order.

0	1	2	3	4
23	32	14	38	7

Let's take an example of bubble sorting order, here we'll compare neighbouring values & check if the 1st value is greater than 2nd, if true we'll swap, if false, will keep as it is & compare next values.



SizeOfArray-1=NoOfRounds.

Here it took 4 rounds to sort this array, If the array is of 100 size then it will take 99 rounds.

Example:

B.java C.java

```
1  package p1;
2      //Sorting an Array
3  public class C {
4      public static void main(String[] args) {
5          int[] x = {23,32,14,38,7};
6
7          for (int i = 0; i < x.length-1; i++) {
8              if(x[i] > x[i+1]); //Checking if the value is greater than 1.
9              int temp = x[i];    //swapping, storing the value of x[i] in temp
10             x[i] = x[i+1];
11             x[i+1] =  temp;
12         }
13     for (int i : x) {
14         System.out.println(i);
15     }
16     }
17 }
```

Console

```
<terminated> C (4) [Java Application] C:\Pr
32
14
38
7
23
```

Here it ran only 1 round, we need to run it four times, for that purpose we're using nested for loop (for loop inside for loop) here.

B.java C.java

```
1  package p1;
2      //Sorting an Array
3  public class C {
4      public static void main(String[] args) {
5          int[] x = {23,32,14,38,7};
6
7          for (int j = 0; j <x.length-1; j++) { //Created one more for loop
8
9              //paste the below for loop block here
10
11
12         }
13         for (int i = 0; i < x.length-1; i++) {
14             if(x[i] > x[i+1]) { //Checking if the value is greater than 1.
15                 int temp = x[i];    //swapping, storing the value of x[i] in temp
16                 x[i] = x[i+1];
17                 x[i+1] =  temp;
18             }
19         }
20     for (int i : x) {
21         System.out.println(i);
22     }
23     }
24 }
```

Console

```
<terminated> C (4) [Java Application] C:\Pr
23
14
32
7
38
```

B.java C.java

```
1  package p1;
2      //Sorting an Array
3  public class C {
4      public static void main(String[] args) {
5          int[] x = {23,32,14,38,7};
6
7          for (int j = 0; j <x.length-1; j++) { //Created one more for loop
8              for (int i = 0; i < x.length-1; i++) {
9                  if(x[i] > x[i+1]) { //Checking if the value is greater than 1.
10                     int temp = x[i];    //swapping, storing the value of x[i] in temp
11                     x[i] = x[i+1];
12                     x[i+1] =  temp;
13                 }
14             }
15         }
16
17     for (int i : x) {
18         System.out.println(i);
19     }
20     }
21 }
```

Console

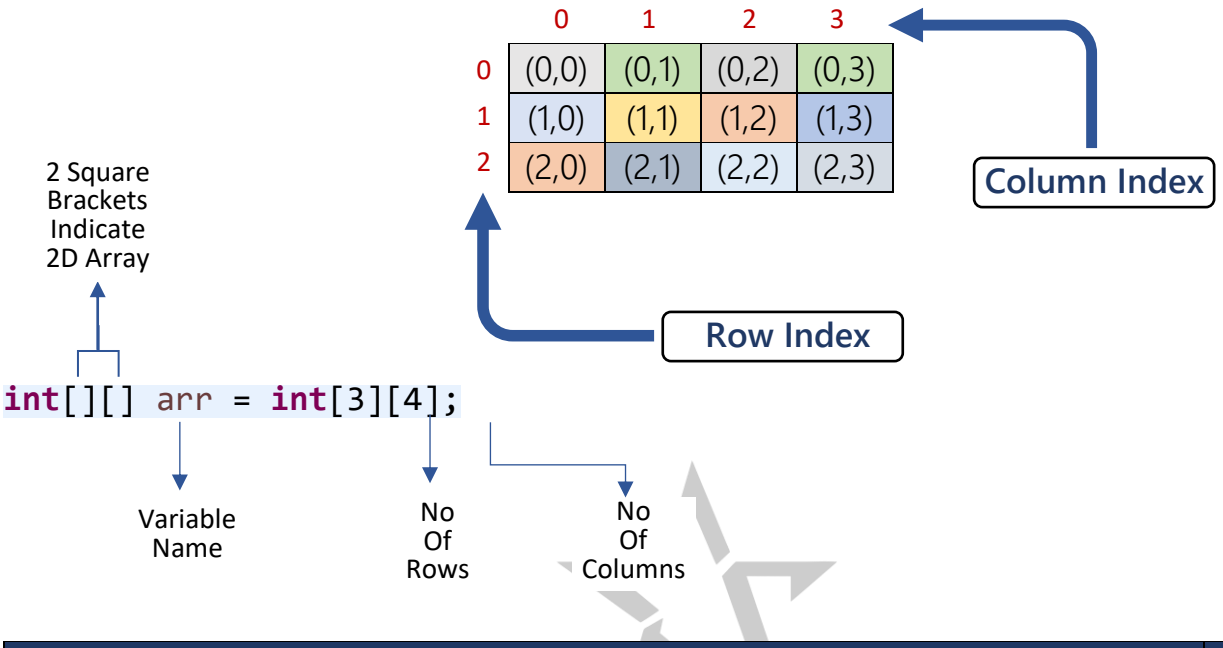
```
<terminated> C (4) [Java Application] C:\Pr
7
14
23
32
38
```

The above array is sorted.

08/06/2022 (Wednesday)

2D – ARRAY

The Two Dimensional Array in Java programming language is nothing but an Array of Arrays. In Java Two Dimensional Array, data stored in row and columns, and we can access the record using both the row index and column index (like an Excel File).



Example	Output

```
package p1;
public class A {
	public static void main(String[] args) {
		int[][] arr = new int[3][4];
		System.out.println(arr.length); //To check no. of rows
		System.out.println(arr[0].length); //To check no. of columns
	}
}
```

```
3
4
```

Example #2:

```
1  package p1;
2  public class A {
3      public static void main(String[] args) {
4          int[][] arr = new int[3][4];
5           //Storing the data
6          arr[0][0] = 10; //0th row, 0th column
7          arr[0][1] = 20; //0th row, 1st column
8          arr[0][2] = 30;
9          arr[0][3] = 40;
10         arr[1][0] = 50;
11         arr[1][1] = 60;
12         arr[1][2] = 70;
13         arr[1][3] = 80;
14         arr[2][0] = 90;
15         arr[2][1] = 100;
16         arr[2][2] = 110;
17         arr[2][3] = 120;
18         for (int i = 0; i < arr.length; i++) {
19             for (int j = 0; j < arr[0].length; j++) {
20                 System.out.println(arr[i][j]);
21             }
22         }
23     }
24 }
```

```
<terminated> A (22) [Java Application]
10
20
30
40
50
60
70
80
90
100
110
120
```

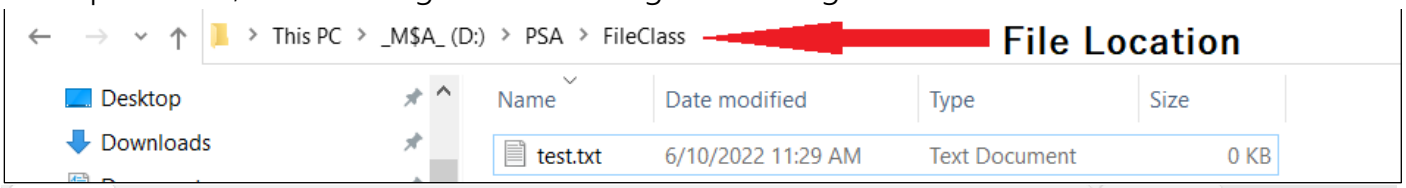
FILE HANDLING IN JAVA

In Java, with the help of File Class, we can work with files. This File Class is inside the java.io package. The File class can be used by creating an object of the class and then specifying the name of the file.

Q. Why File Handling is Required?

A. File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.

In simple words, file handling means reading and writing data to a file.



B.java

```
1 package p1;
2 import java.io.File; //import File Class (create object the, Ctrl+1)
3 public class B {
4     public static void main(String[] args) {
5         File f = new File("D:\\PSA\\FileClass\\test.txt");
6         //It won't check wether file is present or not, it'll directly print path
7         System.out.println(f);
8     }
9 }
```

Console

```
<terminated> B (16) [Java Applicatio
D:\PSA\FileClass\test.txt
```

Exists Method: It is non static method present in File Class, the return type of this method is aboolean value. This method will check whether the file exists in the given path, if the file exists it will true or else false. Example:

B.java

```
1 package p1;
2 import java.io.File;
3 public class B {
4     public static void main(String[] args) {
5         File f = new File("D:\\PSA\\FileClass\\test.txt");
6         f.exists();
7         System.out.println(f.exists());
8     }
9 }
```

Console

```
<terminated> B (16) [Java
true
```

If we want to check the return type, we can also use the exists method like below example:

B.java

```
1 package p1;
2 import java.io.File;
3 public class B {
4     public static void main(String[] args) {
5         File f = new File("D:\\PSA\\FileClass\\test.txt");
6         //To check return type, Ctrl+1 & "Select Assign Statement to new local variable"
7         boolean val = f.exists();
8         System.out.println(val);
9     }
10 }
```

Console

```
<terminated> B (
true
```

Delete Method: It is a non static method present in file class, whose return type is a boolean value, if the file/folder is deleted it will return true or else false. Example:

B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          File f = new File("D:\\PSA\\FileClass\\test.txt");
6          //To check return type, Ctrl+1 & "Select Assign Statement to new local variable"
7          boolean val = f.delete();
8          System.out.println(val);
9      }
10 }
```

Console

```
<terminated> B (16) [Java A
true
```

Create A New File Method: It is a non static method in file class, in the given path if the file is not present, then only it will create a new file & return the boolean value as true, if file already exists then it will not create a file neither override or replace the file & will return as false.

B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          File f = new File("D:\\PSA\\FileClass\\test.txt");
6          boolean val = f.createNewFile();
7          System.out.println(val);
8      }
9  }
```

Console

```
<terminated> B (16) [Java Application]
Exception in thread "main" j
        Unhandled exception

        at p1.B.main(B.java:
```

There's nothing wrong here, still we get CompileTime/Checked Exception, in such situations its's a rule that we should mandatorily surround it with "try catch" block or else the program will not run.

*B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          File f = new File("D:\\PSA\\FileClass\\test.txt");
6          boolean val = f.createNewFile();
7      }
8  }
```

Unhandled exception type IOException
2 quick fixes available:
Add throws declaration
Surround with try/catch
Press 'F2' for focus

Console

```
<terminated> B (16) [Java Application] C:\Program Files\Ja
Exception in thread "main" java.lang.Error:
        Unhandled exception type IOExceptic

        at p1.B.main(B.java:6)
```

B.java

```
1  package p1;
2  import java.io.File;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          File f = new File("D:\\PSA\\FileClass\\test.txt");
7          try {
8              boolean val = f.createNewFile();
9              System.out.println(val);
10         } catch (IOException e) {
11             e.printStackTrace();
12         }
13     }
14 }
```

Console

```
<terminated> B (16) [Java Applicatio
true
```

Make Directory (Folder) Method: It is a non static method present in file class, if the folder doesn't exist in the given path it will create a new folder & return the boolean value as true or else false. Example:

B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          //Folder will not have any extension
6          File f = new File("D:\\PSA\\FileClass\\Test Folder");
7          boolean val = f.mkdir();
8          System.out.println(val);
9      }
10 }
```

Console

```
<terminated> B (16
true
```

Delete Directory (Folder) Method: we use same method "f.delete()" for deleting file as well as folder.

*B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          //Folder doesn't have any extension
6          File f = new File("D:\\PSA\\FileClass\\Test Folder");
7          boolean val = f.delete();
8          System.out.println(val);
9      }
10 }
```

Console

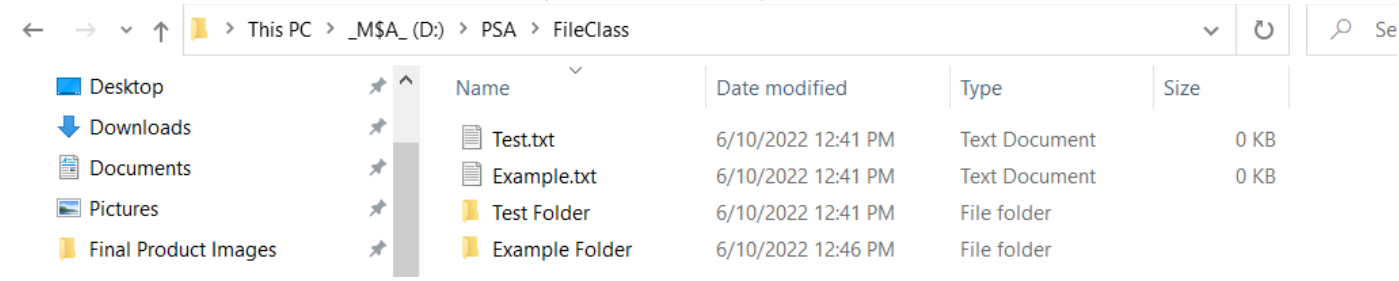
```
<terminated> B (16) [Java
true
```

09/06/2022 (Thursday)

FILE HANDLING IN JAVA (Contd...)

List Method: It's a non static method present in File Class, it will get all the file/folder names present in the given path & the return type of this method is String Array.

Let's first check the Files and folders present in the path:



Example:

B.java

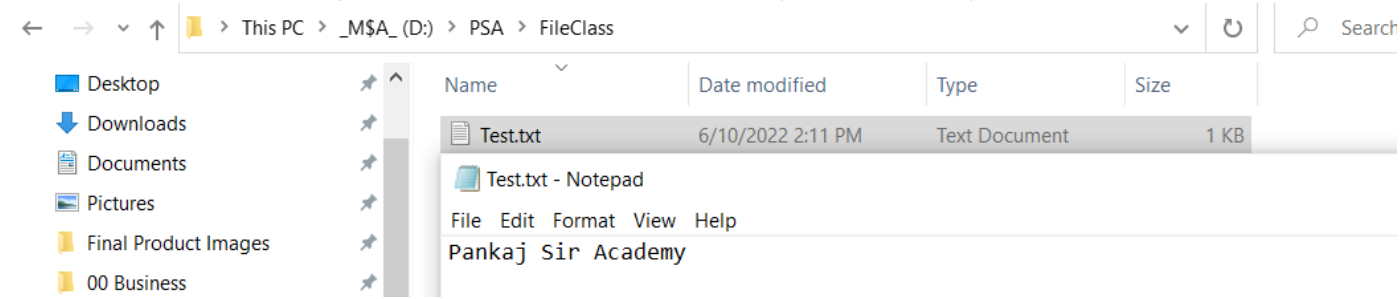
```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          File f = new File("D:\\PSA\\FileClass\\");
6          //to store the list of file names we need to create an array
7          String[] fileNames = f.list();
8          //Let's print the values present in String array
9          for (String s : fileNames) {
10             System.out.println(s);
11         }
12         // to get the count of the files and folders, let's print lenght of array
13         System.out.println(fileNames.length);
14     }
15 }
```

Console

```
<terminated> B (16) [Ja
Example Folder
Example.txt
Test Folder
Test.txt
4
```

Length Method: It's a non static method present in File Class, it counts the number of characters in the given file including spaces.

First let's check what's present in test.text file: "Pankaj Sir Academy" (18 Characters)



Example:

B.java

```
1  package p1;
2  import java.io.File;
3  public class B {
4      public static void main(String[] args) {
5          File f = new File("D:\\PSA\\FileClass\\test.txt");
6          System.out.println(f.length()); //f.length counts the characters
7      }
8  }
```

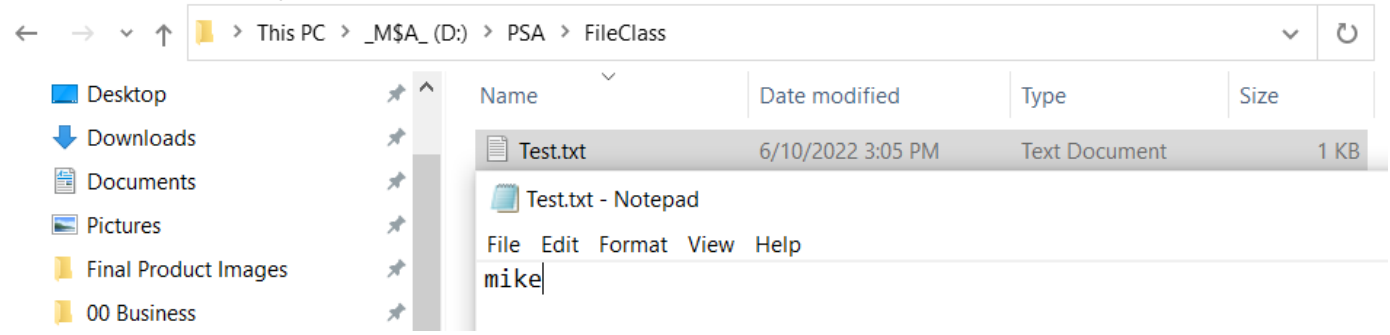
Console

```
<terminated> B (
18
```

FILE READER IN JAVA

Read Method: It's a non static method present in FileReader Class, It is used to read and return a single character in the form of an integer value that contains the character's unicode value, then integer value is converted to char. The FileReader Class doesn't have legnth method, so we use File Class to make the output dynamic.

Let's read the data present in test.txt.




```java
package p1;
import java.io.FileReader;
import java.io.IOException;
public class B {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("D:\\PSA\\FileClass\\test.txt");
            System.out.println(fr.read());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
<terminated> B
109
```

In the above example it read only "m" of "mike" in the integer form, to make it read all the characters we print `System.out.println` = no. of characters present in file.

```java
package p1;
import java.io.FileReader;
import java.io.IOException;
public class B {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("D:\\PSA\\FileClass\\test.txt");
            System.out.println(fr.read());
            System.out.println(fr.read());
            System.out.println(fr.read());
            System.out.println(fr.read());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
<terminated> B
109
105
107
101
```

Let's convert the integer value into character.

B.java

```
1  package p1;
2  import java.io.FileReader;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\test.txt");
8              System.out.println((char)fr.read());
9              System.out.println((char)fr.read());
10             System.out.println((char)fr.read());
11             System.out.println((char)fr.read());
12         } catch (IOException e) {
13             e.printStackTrace();
14         }
15     }
16 }
```

Console

```
<terminated> B
m
i
k
e
```

We cannot increase line of codes to print each character, so here we're using for loop

B.java

```
1  package p1;
2  import java.io.FileReader;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\test.txt");
8              for (int i=0; i<4; i++) { //mike=4 characters
9                  System.out.println((char)fr.read());
10             }
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14     }
15 }
```

Console

```
<terminated> B
m
i
k
e
```

To get the character values in a single line, let's remove "ln" from `System.out.println`

*B.java

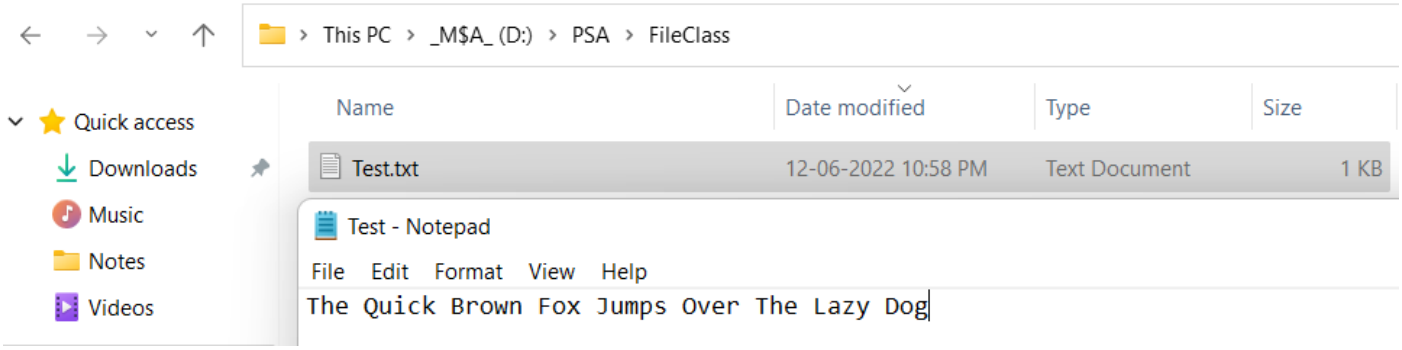
```
1  package p1;
2  import java.io.FileReader;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\test.txt");
8              for (int i=0; i<4; i++) { //mike=4 characters
9                  System.out.print((char)fr.read()); //remove "ln" to print values in single line
10             }
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14     }
15 }
```

Console

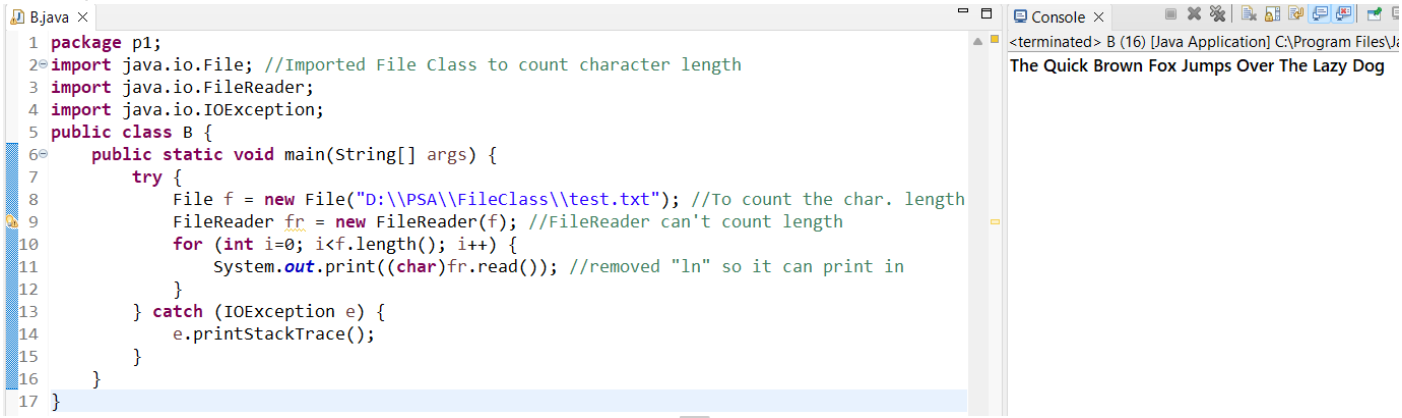
```
<terminated> B
mike
```

Here in for loop condition (`for (int i=0; i<4; i++) {`) we gave i<4, because mike has 4 characters & we want the for loop to run till 3 which is less than 4. But we cannot keep giving the condition based upon the character length, & FileReader Class cannot count the length, to automate this we'll use File Class to count the length of the characters. Example:

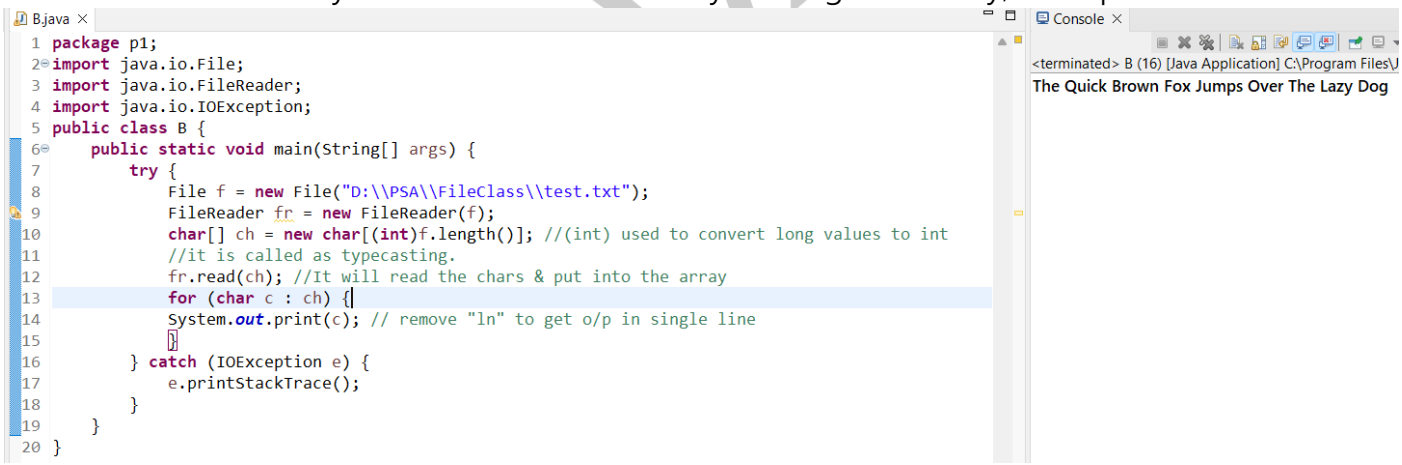
The character present in test.txt file:



Example:

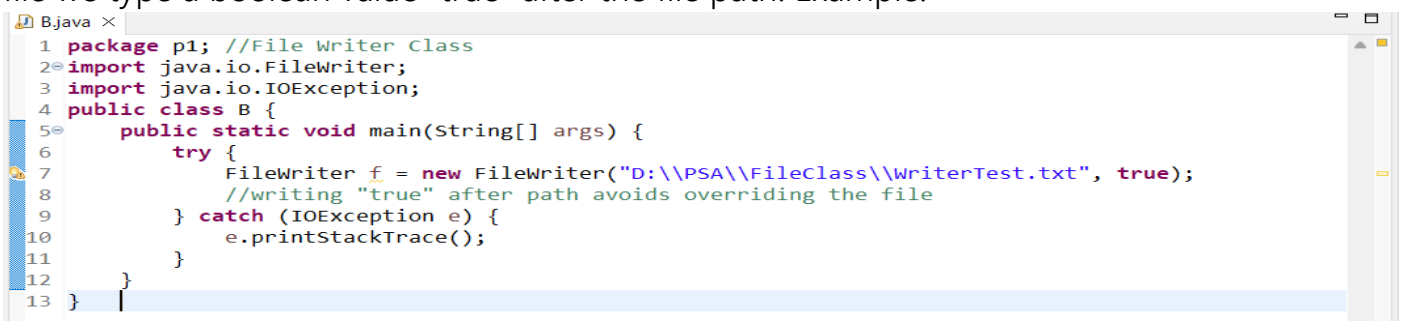


There's an another way to to read the text file by making char array, example:



FILE WRITER IN JAVA

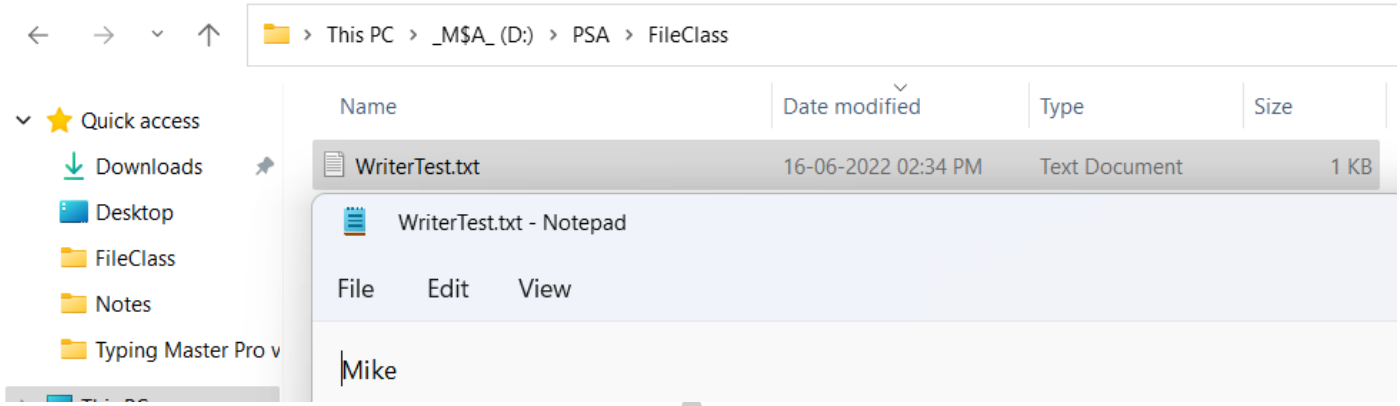
Write Method: It's a non static method present in FileWriter Class, by default it will override the existing file or if the file doesn't exists it will create a new file. To avoid overriding of the file we type a boolean value "true" after the file path. Example:



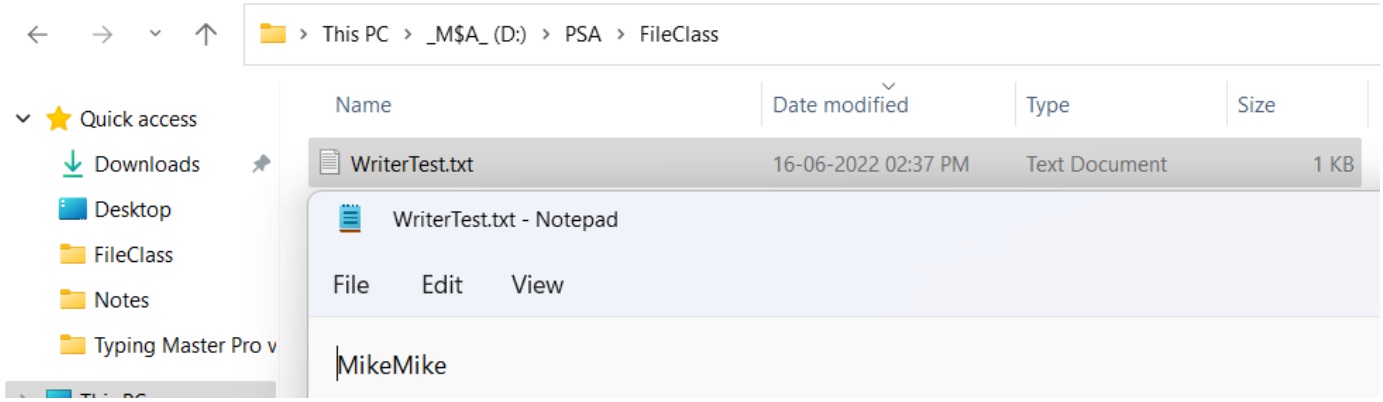
Let's write "mike" into the file, after writing always use close() method to save and close the file, if we don't close it, it wont write anything in the file. Example:

```java
package p1; //File Writer Class
import java.io.FileWriter;
import java.io.IOException;
public class B {
    public static void main(String[] args) {
        try {
            FileWriter f = new FileWriter("D:\\PSA\\FileClass\\WriterTest.txt", true);
            f.write("Mike");
            f.close(); //It saves the file and close it.
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
Console
<terminated> B (16) [Java Application] C:\Progr
```

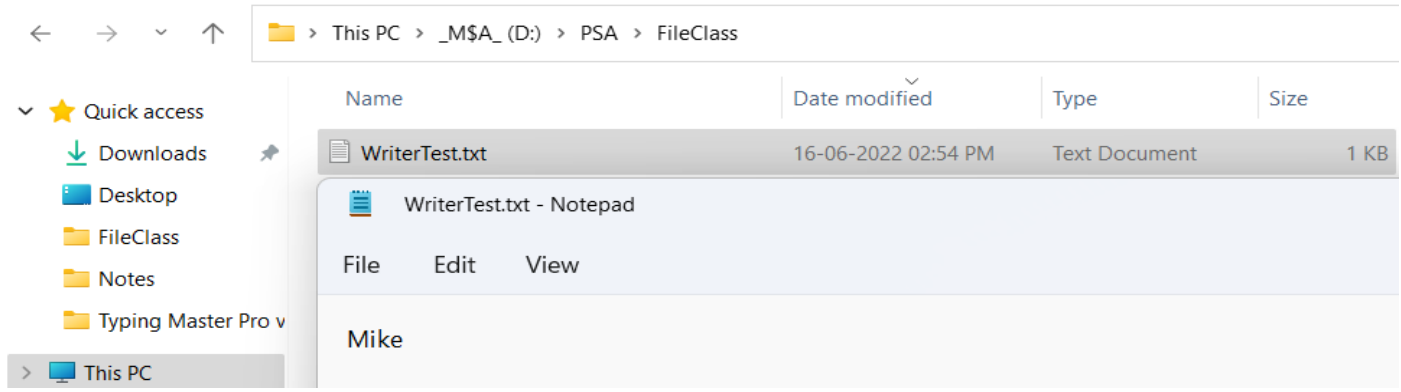


Let's run the code again:



It wrote the "Mike" once again without overriding the file, due to the "true" written after path. Now we're writing the "false" after path, Example:

```java
package p1; //File Writer Class
import java.io.FileWriter;
import java.io.IOException;
public class B {
    public static void main(String[] args) {
        try {
            FileWriter f = new FileWriter("D:\\PSA\\FileClass\\WriterTest.txt", false);
            f.write("Mike");
            f.close(); //It saves the file and close it.
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

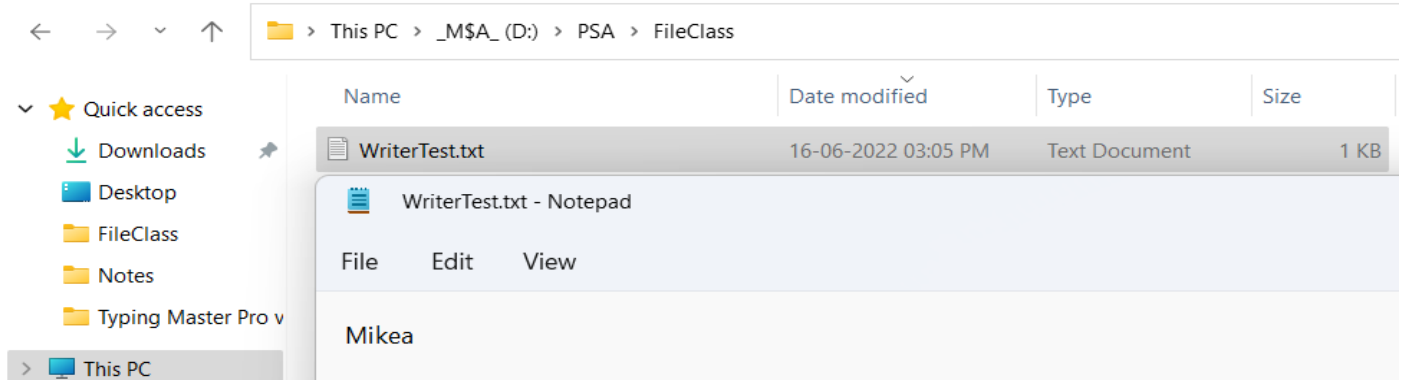


It replaced (overrode) the file with new one.

Example: *(writing int value to the file without double quotes)*

B.java

```
1  package p1; //File Writer Class
2  import java.io.FileWriter;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileWriter f = new FileWriter("D:\\PSA\\FileClass\\WriterTest.txt", false);
8              f.write("Mike");
9              f.write(97);
10             f.close(); //It saves the file and close it.
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14     }
15 }
```



It wrote 97's coreesponding char value "a" after "Mike", so always write anything (int, char, boolean etc..) in double quotes "" (String value) to write into the text files.

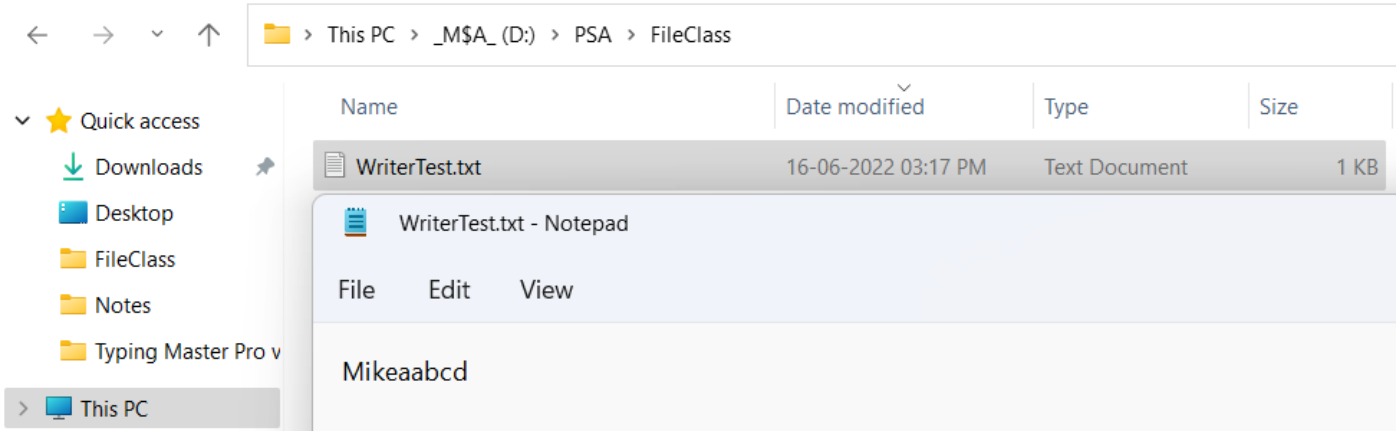
Another way to write into the file. Example:

*B.java

```
1  package p1; //File Writer Class
2  import java.io.FileWriter;
3  import java.io.IOException;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileWriter f = new FileWriter("D:\\PSA\\FileClass\\WriterTest.txt", false);
8              f.write("Mike");
9              f.write(97);
10             //another method to write
11             char[] x = {'a', 'b', 'c', 'd'};
12             f.write(x);
13             f.close();
14         } catch (IOException e) {
15             e.printStackTrace();
16         }
17     }
18 }
```

Console

<terminated> B (16



Task Assignment

Watch recorded lectures from Pankaj Sir Academy App

Batch > Recorded Series Part 1
Lectures: **28/10/2020** to **04-11-2020** (9 videos)

Batch > Recorded Series Part 2
Lectures: **05/11/2020** to **20-11-2020** (18 videos)

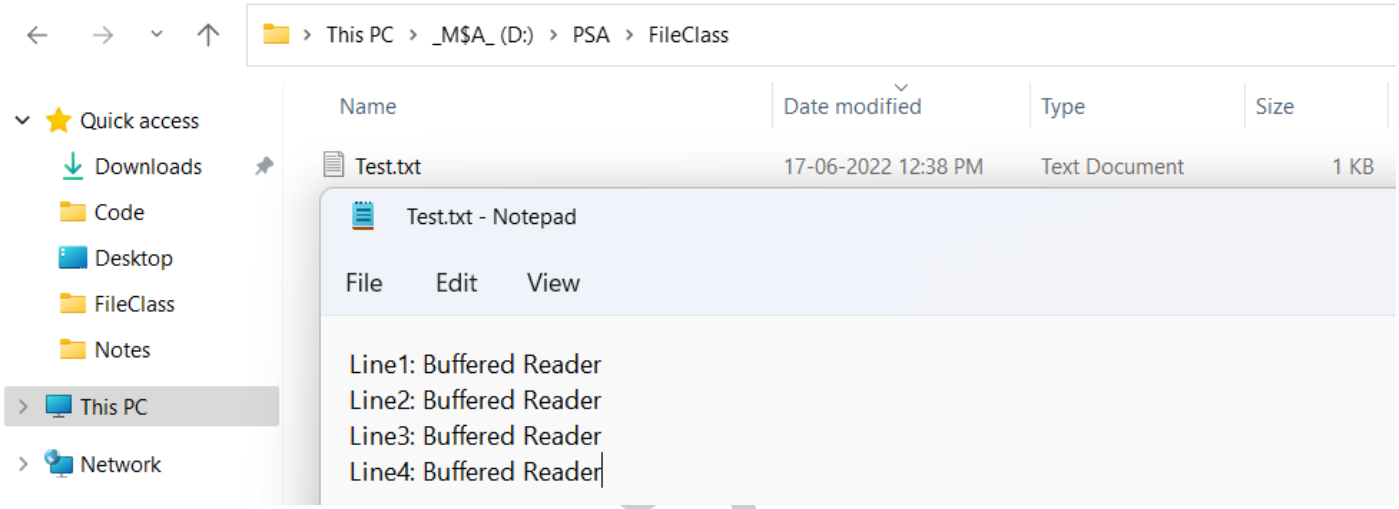
10/06/2022 (Friday)

BUFFERED READER AND WRITER IN JAVA

Buffered Reader:

- It improves file reading performance
- It has an exclusive readLine() method that reads the content line by line.

File Content:



Example:

A.java

```
1  package p1;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  public class A {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\Test.txt");
8              //Without FileReader Class BufferedReader can't work
9              BufferedReader br = new BufferedReader(fr);
10             System.out.println(br.readLine());
11         } catch (Exception e) {
12             e.printStackTrace();
13         }
14     }
15 }
```

Console

```
<terminated> A (23) [Java Ap
Line1: Buffered Reader
```

Example #2:

A.java B.java

```
1  package p1;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  public class A {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\Test.txt");
8              //Without FileReader Class BufferedReader can't work
9              BufferedReader br = new BufferedReader(fr);
10             System.out.println(br.readLine()); //reads 1st line
11             System.out.println(br.readLine()); //reads 2nd line
12             System.out.println(br.readLine()); //reads 3rd line
13         } catch (Exception e) {
14             e.printStackTrace();
15         }
16     }
17 }
```

Console

```
<terminated> A (23) [Java Ap
Line1: Buffered Reader
Line2: Buffered Reader
Line3: Buffered Reader
```

Example #3: *(To read all the content from a text file, using while loop)*

A.java B.java C.java

```
1  package p1;
2  import java.io.BufferedReader;
3  import java.io.FileReader;
4  public class C {
5      public static void main(String[] args) {
6          try {
7              FileReader fr = new FileReader("D:\\PSA\\FileClass\\Test.txt");
8              BufferedReader br = new BufferedReader(fr);
9              //To read all the content from a text file, using while loop
10                int i;
11                while((i=br.read())!=-1){
12                System.out.print((char)i);
13                }
14         } catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18 }
```

Console

```
<terminated> C (6) [Java Appli
Line1: Buffered Reader
Line2: Buffered Reader
Line3: Buffered Reader
Line4: Buffered Reader
Line5: Buffered Reader
Line6: Buffered Reader
Line7: Buffered Reader
Line8: Buffered Reader
Line9: Buffered Reader
Line10: Buffered Reader
```

Buffered Writer:

- It improves file writing performance
- It has an exclusive newLine() method that writes the content in new line.

Example:

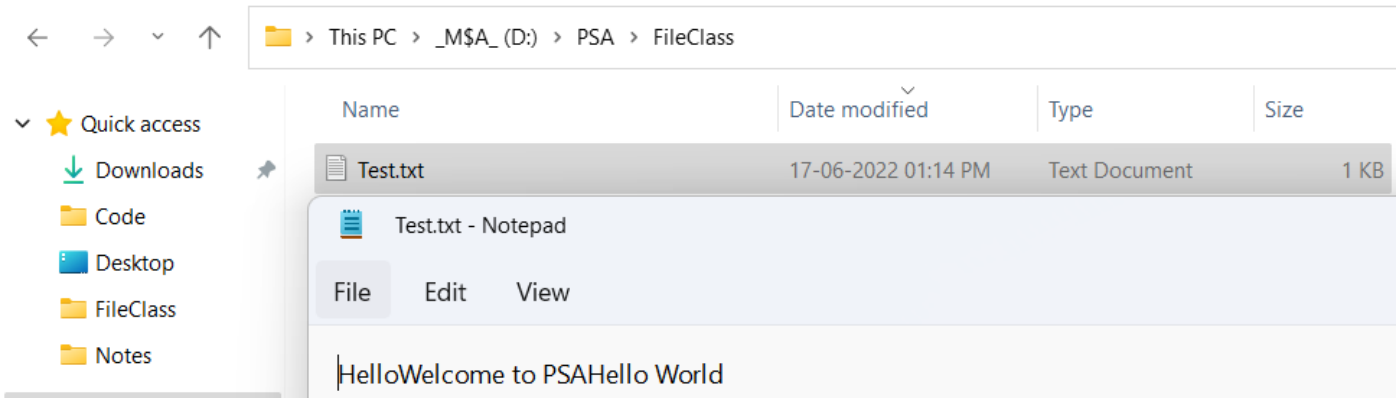
B.java

```
1  package p1;
2  import java.io.BufferedWriter;
3  import java.io.FileWriter;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileWriter fw = new FileWriter("D:\\PSA\\FileClass\\Test.txt");
8              //Without FileWriter Class BufferedWriter can't work
9              BufferedWriter bw = new BufferedWriter(fw);
10             bw.write("Hello");
11             bw.write("Welcome to PSA");
12             bw.write("Hello World");
13             bw.close();
14             fw.close();
15         } catch (Exception e) {
16             e.printStackTrace();
17         }
18     }
19 }
```

Console

```
<terminated> B (17) [
```

Output:



To write in a new line we insert newLine() method. Example:

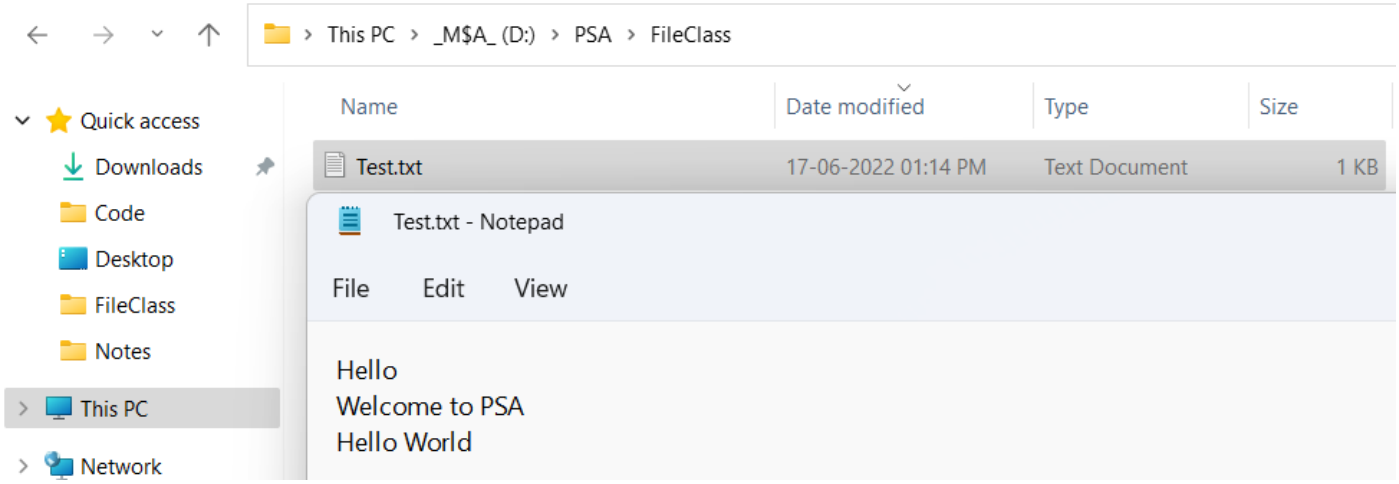
B.java

```java
package p1;
import java.io.BufferedWriter;
import java.io.FileWriter;
public class B {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("D:\\PSA\\FileClass\\Test.txt");
            //Without FileWriter Class BufferedWriter can't work
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("Hello");
            bw.newLine(); //To insert new line
            bw.write("Welcome to PSA");
            bw.newLine();
            bw.write("Hello World");
            bw.close();
            fw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Console

<terminated> B (17) [Ja

Output:



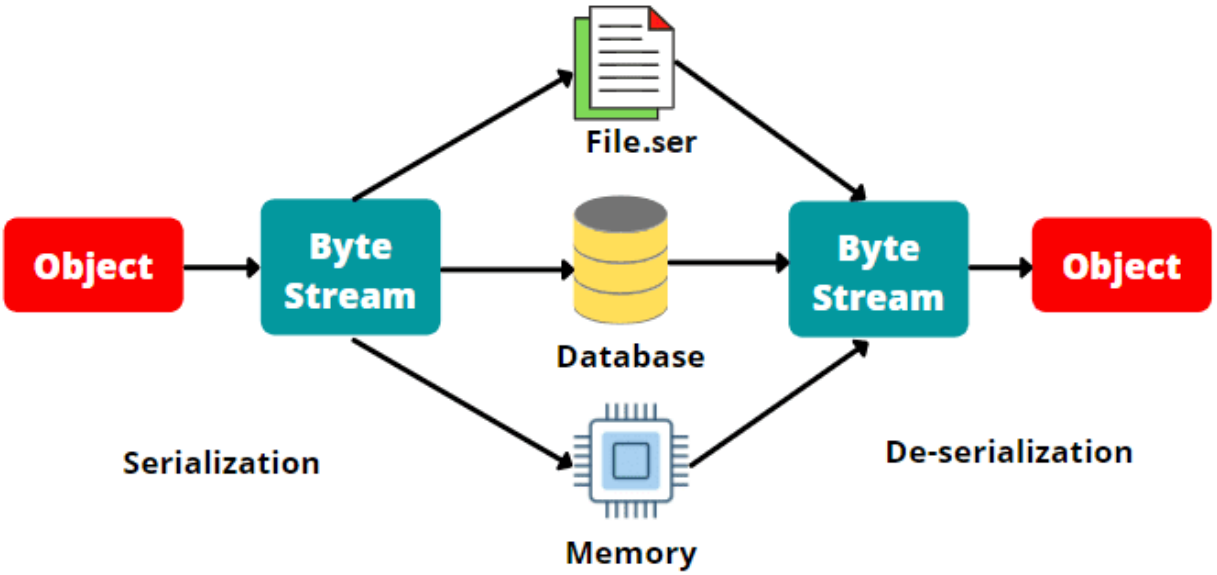
Q. What is Serialization and De-Serialization.?

SERIALIZATION AND DE-SERIALIZATION

All the objects we create in java are stored in RAM (Heap Memory) which is a temporary memory, to store it permanently in .ser file we use Serialization and De-Serialization.

In serialization we convert the object in zeros and ones and then store the object state in permanantly in file.

In de-serialization we read binaries from the file and reconstruct the object back.

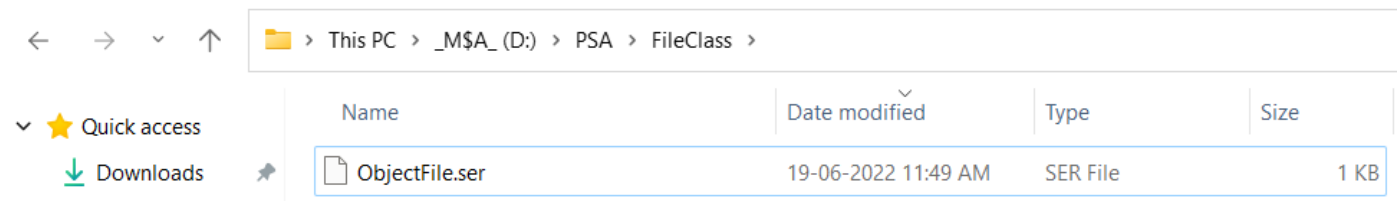


Example:

A.java
```
1  package p2;
2  import java.io.Serializable; //Serializable Class imported
3  public class A implements Serializable { //Marker Interface (Empty Interface)
4      //Serializable interface lets the compiler to serialise the class objects
5      String name = "Sayyed";
6      String city = "Hyderabad";
7  }
```

B.java
```
1  package p2;
2  import java.io.FileOutputStream;
3  import java.io.ObjectOutputStream;
4  public class B {
5      public static void main(String[] args) {
6          try {
7              FileOutputStream fos = new FileOutputStream("D:\\PSA\\FileClass\\ObjectFile.ser");
8              ObjectOutputStream oos = new ObjectOutputStream(fos);
9              A a1 = new A();
10             oos.writeObject(a1);
11             oos.close();
12             fos.close();
13         } catch (Exception e) {
14             e.printStackTrace();
15         }
16     }
17 }
```

OutPut:



Q. What is marker interface?

A. An empty interface is called as Marker Interface.

13/06/2022 (Monday)

DE-SERIALIZATION

Q. What is ObjectClass in JAVA?

A. The super most class in JAVA is called as ObjectClass in JAVA.

Example:

```java
package p2; //De-Serialization
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class C {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("D:\\PSA\\FileClass\\ObjectFile.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            A a1 = (A) ois.readObject(); //Typecast & Object (instance) of type A
            System.out.println(a1.name);
            System.out.println(a1.city);
            ois.close();
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Console
<terminated>
Sayyed
Hyderabad
```

"transient" (keyword):

During Serialization "transient" keyword when applied on a variable, it will skip writing into the object.

Example:

```java
package p2;
import java.io.Serializable; //Serializable Class imported
public class A implements Serializable { //Marker Interface (Empty Interface)
    //Serializable interface lets the compiler to serialise the class objects
    String name = "Sayyed";
    String city = "Hyderabad";
    transient String password = "Test1234";
}
```

```java
package p2; //De-Serialization
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class C {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("D:\\PSA\\FileClass\\ObjectFile.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            A a1 = (A) ois.readObject(); //Typecast & Object (instance) of type A
            System.out.println(a1.name);
            System.out.println(a1.city);
            System.out.println(a1.password); //Transient
            ois.close();
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Console
<terminated> C
Sayyed
Hyderabad
null
```

JAVA DATA BASE CONNECTIVITY (JDBC)

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. JDBC API uses JDBC drivers to connect with the database.
We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.
There are lot of database softwares, but most popular ones are:

We can use any one, but here we are using MySQL.

Example#1: *(we're creating one Registration table in MySQL to store the data)*

```
create my_db
use my_db
create table Registration
(
   Name varchar(45),
   City varchar(45),
   Email varchar(128),
   MobileNumber varchar(10)
)
select * from Registration
-- * means select all the columns in the table, we can also give column name
insert into Registration values('Sayyed', 'Hyderabad', '672msa@gmail.com', '1234567890')
-- I ran it twice, so it saved the data twice in the Registration table
insert into Registration values('Mike', 'Chennai', 'mike@gmail.com', '0123456789')
insert into Registration values('Michael', 'Tamil Nadu', 'michael@gmail.com', '1111111111')
insert into Registration values('Banty Bablu', 'Kerala', 'banty@gmail.com', '2222222222')
```

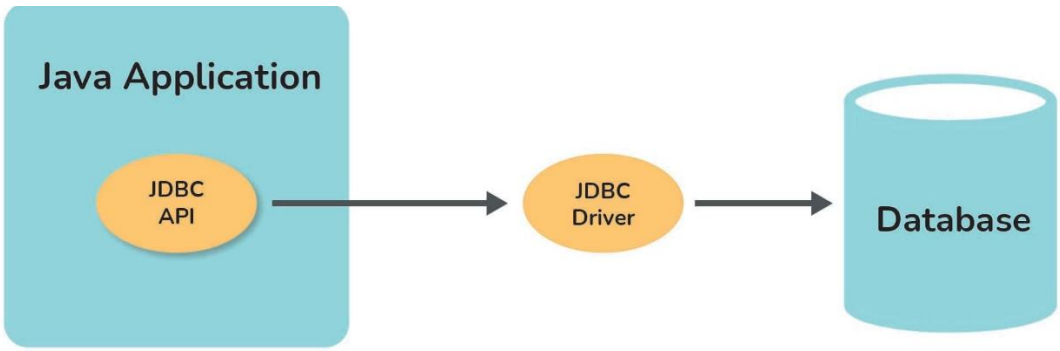
Result Grid:

Name	City	Email	MobileNumber
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Mike	Chennai	mike@gmail.com	0123456789
Michael	Tamil Nadu	michael@gmail.com	1111111111
Banty Bablu	Kerala	banty@gmail.com	2222222222

14/06/2022 (Tuesday)

JAVA DATABASE CONNECTIVITY – JDBC (Contd...)

To connect to the database we require a JDBC Driver. The JDBC Driver is a software component that enables java application to interact with the database.



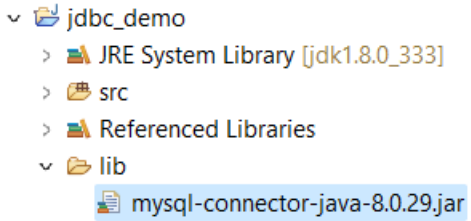
There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

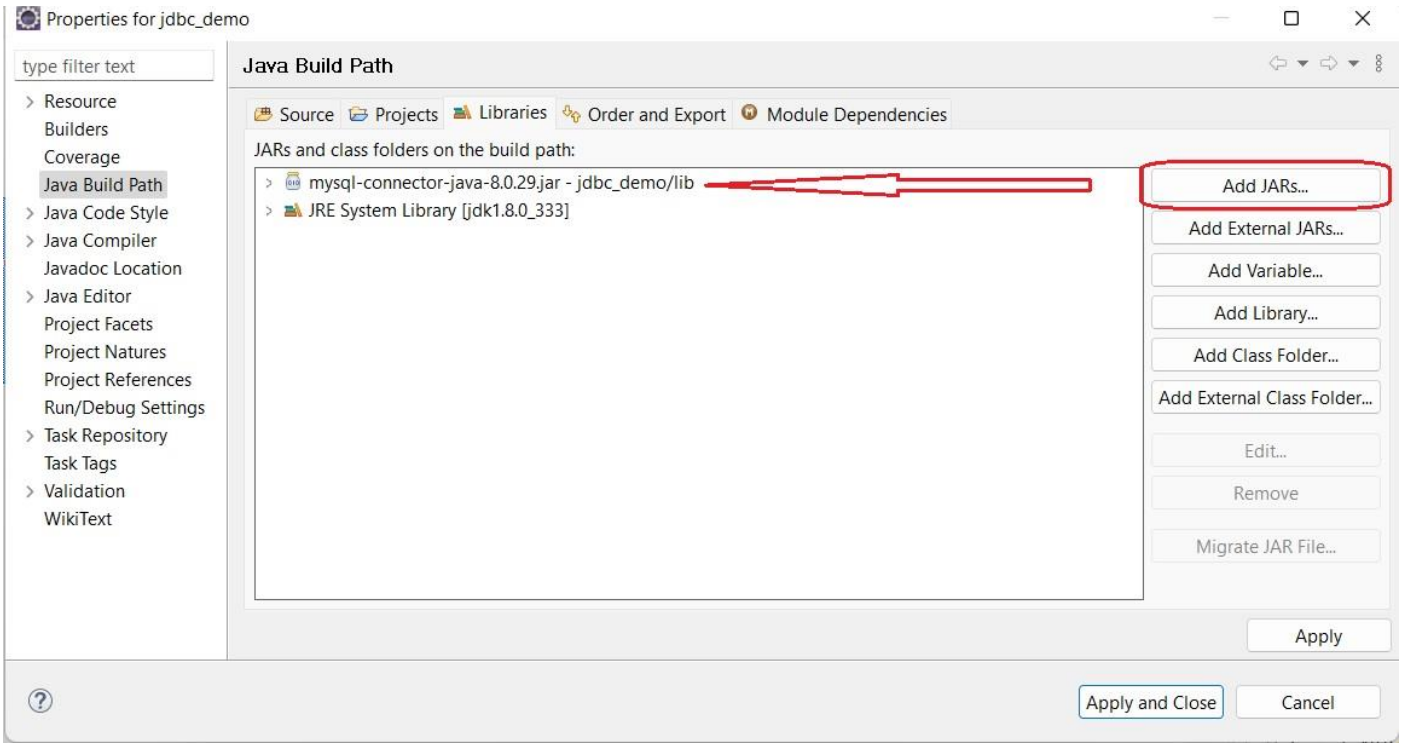
To connect to our MySQL database, we are using a JDBC Driver called "MySQL Connector/J", MySQL Connector/J is a JDBC Type 4 driver, the Type 4 designation means that the driver is a pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.

Download Link for MySQL Connector/J:
https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.29.zip

After extracting the .zip file we're copying the "mysql-connector-java-8.0.29.jar" file into our JAVA Project.
Create a new folder in JAVA Project named "lib" & paste the "mysql-connector-java-8.0.29.jar" into that folder.



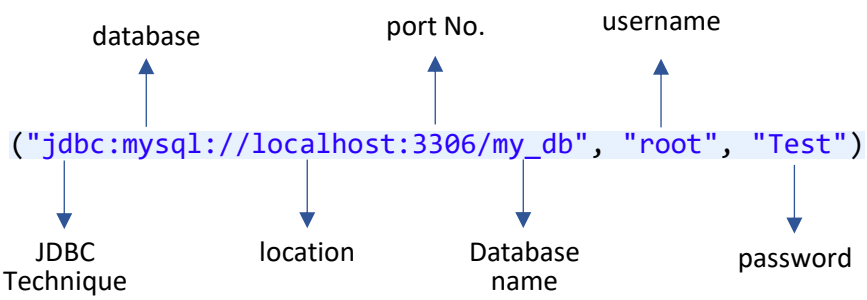
After pasting configure the .jar file to the Java project, to do that go to:
JAVAProject>Properties>Java Build Path>Libraries>Add JARs>Select "mysql-connector-java-8.0.29.jar".



Example #1: JDBC code to connect with MySQL database:

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager; //It's built in method to connect to database
//make java code connect to database
public class A {
	public static void main(String[] args) {
		try {
			//3 pararameters to connect to database. URL, UserName, Password
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:
com.mysql.cj.jdbc.ConnectionImpl@780cb77



Example #2: Code to store/insert record in database:

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class A {
	public static void main(String[] args) {
		try {

			//Make java code connect to database:
			Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

			//Code to store/insert record into database:
			Statement st = con.createStatement();
			st.executeUpdate
("insert into Registration values('Stallin', 'Chicago', 'stallin@gmail.com', '1084658760')");

			//close database connection
			con.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:

com.mysql.cj.jdbc.ConnectionImpl@780cb77

Result Grid in MySQL:

Name	City	Email	MobileNumber
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Mike	Chennai	mike@gmail.com	0123456789
Michael	Tamil Nadu	michael@gmail.com	1111111111
Banty Bablu	Kerala	banty@gmail.com	2222222222
Stallin	Chicago	stallin@gmail.com	1084658760

Example #3: Code to delete the record from database:

We can delete record based on Name, City, Email, MobileNumber, here we're using deletion based on Email, because Email and Mobile Numbers are always unique, but if we delete based on Name, City there might be multiple entries with the Name/City, all will be deleted.

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class A {
	public static void main(String[] args) {
		try {

			//Make java code connect to database:
			Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

			//Code to delete record from database: deleting Mike's Data
			Statement st = con.createStatement();
			st.executeUpdate
("DELETE FROM Registration WHERE Email='mike@gmail.com'");

			//close database connection
			con.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:

com.mysql.cj.jdbc.ConnectionImpl@780cb77

Result Grid in MySQL:

Name	City	Email	MobileNumber
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Michael	Tamil Nadu	michael@gmail.com	1111111111
Banty Bablu	Kerala	banty@gmail.com	2222222222
Stallin	Chicago	stallin@gmail.com	1084658760

Example #4: Code to update the record in database:

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
public class A {
	public static void main(String[] args) {
		try {

			//Make java code connect to database:
			Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

			//Code to update in database: updating Stalli's City from "Chicago" to "NYC"
		Statement st = con.createStatement();
			st.executeUpdate
("UPDATE Registration SET City = 'NYC' WHERE email='stallin@gmail.com'");

			//close database connection
			con.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:

com.mysql.cj.jdbc.ConnectionImpl@780cb77

Result Grid in MySQL:

Name	City	Email	MobileNumber
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Sayyed	Hyderabad	672msa@gmail.com	1234567890
Michael	Tamil Nadu	michael@gmail.com	1111111111
Banty_Bablu	Kerala	banty@gmail.com	2222222222
Stallin	NYC	stallin@gmail.com	1084658760

15/06/2022 (Wednesday)

JAVA DATABASE CONNECTIVITY – JDBC (Contd...)

Example #5: Code to read the data in database:

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class D {
	public static void main(String[] args) {
		try {

		//Make java code connect to database:
			Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

		//Code to read the data in database: (we use executeQuery in place of executeUpdate
			Statement st = con.createStatement();
			ResultSet result = st.executeQuery
("select * from Registration"); //collection of values

			while(result.next()) {
			System.out.println(result.getString(1)); //Column#1 = Name's column
			System.out.println(result.getString(2)); //Column#2 = City's column
			System.out.println(result.getString(3)); //Column#3 = Email's column
			System.out.println(result.getString(4)); //Column#4 = MobileNumber's column
			}

		//close database connection
			con.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:

```
com.mysql.cj.jdbc.ConnectionImpl@780cb77
Sayyed
Hyderabad
672msa@gmail.com
1234567890
Sayyed
Hyderabad
672msa@gmail.com
1234567890
Michael
Tamil Nadu
michael@gmail.com
1111111111
Banty Bablu
Kerala
banty@gmail.com
2222222222
Stallin
NYC
stallin@gmail.com
1084658760
```

Getting the output of the above code in an organised way:

```
package jdbc_demo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class D {
	public static void main(String[] args) {
		try {

		//Make java code connect to database:
			Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

		//Code to read the data in database: (we use executeQuery in place of executeUpdate
			Statement st = con.createStatement();
			ResultSet result = st.executeQuery
("select * from Registration"); //collection of values

			while(result.next()) {
			System.out.print(result.getString(1) + "," + " "); //Column#1 = Name's column
			System.out.print(result.getString(2) + "," + " "); //Column#2 = City's column
			System.out.print(result.getString(3) + "," + " "); //Column#3 = Email's column
			System.out.print(result.getString(4) + " "); //Column#4 = MobileNumber's column
			System.out.println(""); //just to get the output in next line

		//close database connection
			con.close();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Output:

```
com.mysql.cj.jdbc.ConnectionImpl@780cb77
Sayyed, Hyderabad, 672msa@gmail.com, 1234567890
Sayyed, Hyderabad, 672msa@gmail.com, 1234567890
Michael, Tamil Nadu, michael@gmail.com, 1111111111
Banty Bablu, Kerala, banty@gmail.com, 2222222222
Stallin, NYC, stallin@gmail.com, 1084658760
```

Task Assignment

Watch recorded JDBC lecture from Pankaj Sir Academy App

Batch > Recorded Series Part 2

Lectures: **10/11/2021** to **11-11-2021**

To cover "**.jar file concept**"

CRUD OPERATIONS

As a developer every application involves CRUD Operations.

Q. What is CRUD Operations?

A. The abbreviation CRUD stands for Create, Read, Update, and Delete in computer programming.

In MySQL we use "executeUpdate()" method to Create, Update and Delete the data in database, and "executeQuery()" method to read the data from the database.

MULTIPLE CATCH BLOCK IN TRY-CATCH BLOCK

Q. Can we create more than one catch block in Try-Catch?

A. Yes, we can create multi catch blocks but we should always start with child class exceptions followed by parent class, the matching catch block will be called first, if none of the catch block matches with the exception then the parent class exception automatically handles it.

Note: In place of Exception class, we can also give its parent class Throwable, it can also handle all kinds of Exceptions.

Example #1: *(Child first then parent class exception, No Error)*

A.java

```
1  package p1; //Multiple catch block in try-catch
2  public class A {
3      public static void main(String[] args) {
4          try {
5
6          } catch (ArithmeticException e) {   //ArithmeticException is Child Class Exception
7
8          } catch (NullPointerException e) {  //NullPointerException is child class Exception
9
10         } catch (Exception e) {             //Exception is a parent class
11
12         }
13     }
14 }
```

Example #2: *(parent class first then child exception, Error)*

Error Message: Unreachable catch block for NullPointerException. It is already handled by the catch block for Exception

A.java

```
1  package p1; //Multiple catch block in try-catch
2  public class A {
3      public static void main(String[] args) {
4          try {
5
6          } catch (Exception e) {              //Exception is a parent class
7
8          } catch (NullPointerException e) {  //NullPointerException is child class Exception
9
10         } catch (NullPointerException e) {  //NullPointerException is child class Exception
11
12         }
13     }
14 }
```

Example #3: *(Creating ArithmeticException)*

```
package p1; //Multiple catch block in try-catch
public class A {
	public static void main(String[] args) {
		try {
			int x = 10/0; //It's an arithmetic exception
		} catch (ArithmeticException e) {
			System.out.println("This result is from ArithmeticException");
		} catch (NullPointerException e) {
			System.out.println("This result is from NullPointerException");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		}
	}
}
```

Output:
This result is from ArithmeticException

Example #4: *(Creating NullPointerException)*

```
package p1; //Multiple catch block in try-catch
public class A {
int x =10;
	public static void main(String[] args) {
		try {
			A a1 = null;
			System.out.println(a1.x); //It's a NullPointerException, control goes to Line#10
		} catch (ArithmeticException e) {
			System.out.println("This result is from ArithmeticException");
		} catch (NullPointerException e) {
			System.out.println("This result is from NullPointerException");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		}
	}
}
```

Output:
This result is from NullPointerException

Example #5: *(Creating ArithmeticException and NullPointerException both)*

```
package p1; //Multiple catch block in try-catch
public class A {
int x =10;
	public static void main(String[] args) {
		try {
			int x =10/0; //It's an arithmetic exception
			A a1 = null;
			System.out.println(a1.x); //It's a NullPointerException
		} catch (ArithmeticException e) {
			System.out.println("This result is from ArithmeticException");
		} catch (NullPointerException e) {
			System.out.println("This result is from NullPointerException");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		}
	}
}
```

Output:
This result is from ArithmeticException

Here it handled only ArithmeticException exception, whichever may come first it will handle that exception and stops & will not run further codes.

Example #6: *(Interchanging the exceptions from previous program)*

```
package p1; //Multiple catch block in try-catch
public class A {
int x =10;
	public static void main(String[] args) {
		try {
			A a1 = null;
			System.out.println(a1.x); //It's a NullPointerException
			int x =10/0; //It's an arithmetic exception
		} catch (ArithmeticException e) {
			System.out.println("This result is from ArithmeticException");
		} catch (NullPointerException e) {
			System.out.println("This result is from NullPointerException");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		}
	}
}
```

Output:
This result is from NullPointerException

Here it handled only NullPointerException, because it came first & did not run any further code of blocks.

Example #7: *(creating NumberFormatException wich cannot be handled by ArithmeticException and NullPointerException & control goes to parent class Exception)*

```
package p1; //Multiple catch block in try-catch
public class A {
int x =10;
	public static void main(String[] args) {
		try {
			Integer.parseInt("abcdef"); //It's a NumberFormatException
			A a1 = null;
			System.out.println(a1.x); //It's a NullPointerException
			int x =10/0; //It's an arithmetic exception
		} catch (ArithmeticException e) {
			System.out.println("This result is from ArithmeticException");
		} catch (NullPointerException e) {
			System.out.println("This result is from NullPointerException");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		}
	}
}
```

Output:
This result is from Exception

FINALLY BLOCK

Q. What is Finally Block?

A. Finally block is an extension of Try-Catch. Wether exception occurs or not Finally Block will continue to excecute.

Example #1: *(Exception ocuurs)*

```
package p1; //Finally Block
public class B {
	public static void main(String[] args) {
		try {
			Integer.parseInt("abcdef"); //It's a NumberFormatException
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		} finally {
			System.out.println("This result is from final block");
		}
	}
}
```

Output:
This result is from Exception
This result is from final block

Example #2: *(No Exception)*

```
package p1; //Finally Block
public class B {
	public static void main(String[] args) {
		try {
			Integer.parseInt("100");
		} catch (Exception e) {
			System.out.println("This result is from Exception");
		} finally {
			System.out.println("This result is from finally block");
		}
	}
}
```

Output:
This result is from finally block

Q. Can we write only Try and Finally & skip catch block?

A. Yes, We can write it.

Example #3:

A.java | B.java

```
1  package p1; //Finally Block
2  public class B {
3      public static void main(String[] args) {
4          try {
5              Integer.parseInt("abcdef"); //Exception here
6          } finally {
7              System.out.println("This result is from finally block");
8          }
9          System.out.println("This result is from post finally block");
10     }
11 }
```

Console

```
<terminated> B (19) [Java Application]
This result is from finally block
Exception in thread "main" java.la
	at java.lang.NumberFormatE
	at java.lang.Integer.parseInt(
	at java.lang.Integer.parseInt(
	at p1.B.main(B.java:5)
```

Here exception is happening, but if catch block is missing, exception is not handled so line#9 will not run but finally block continues to run.

Task Assignment

Watch recorded lecture from
Pankaj Sir Academy App
To learn **"Finalize" Keyword.**
Batch > Recorded Series Part 2

Q. What is the difference between Final, Finally and Finalize?

Sr#	Key	final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

Q. Give a practical example where finally block can be used?

A. The suitable example for this would be JDBC, where we Open the connection to the database & an excpetion occurs while typing the code to the database & the closing connection (Close() method) will not run further and the connection remains open.

```
package p1;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class D {
	public static void main(String[] args) {
		Connection con = null; //it was local to try, now it's local to Main
		try {

			//Open Connection
	con = DriverManager.getConnection("jdbc:mysql://localhost:3306/my_db", "root", "Test");
			System.out.println(con);

			//Code to store/insert record into database:
			Statement st = con.createStatement();
			//in below statement there's an exception in SQL Query "inserts"
			st.executeUpdate
("inserts into Registration values('Stallin', 'Chicago', 'stallin@gmail.com', '1084658760')");

			//Close connection
		} catch (Exception e) {
			e.printStackTrace();

		} finally {
			try {
				System.out.println("This result is from finally block");
				con.close();
			} catch (SQLException e1) {
				e1.printStackTrace();
			}
		}
	}
}
```

Output:

java.sql.SQLException: No suitable driver found for jdbc:mysql://localhost:3306/my_db
	at java.sql.DriverManager.getConnection(DriverManager.java:689)
	at java.sql.DriverManager.getConnection(DriverManager.java:247)
	at p1.D.main(D.java:12)
This result is from finally block
Exception in thread "main" java.lang.NullPointerException
	at p1.D.main(D.java:27)

Task Assignment

Watch recorded lecture from
Pankaj Sir Academy App
To learn **"Typecasting"**.
Batch > Recorded Series Part 1

REMAINING TOPICS TO BE COVERED LATER

- Stream API (JAVA 8 Feature)
- Optional Class (JAVA 8 Feature)
- Logical Questions on Collection
- Difference between String buffer and String Builder Class
- Singleton Design Pattern
- Vectors
- Priority Queue
- Sorting an Object
- Difference between Comparator and Comparable Interface
- Runtime Polymorphism