DATE: 22/04/24

## Super Keyword: (Immediate Parent Class)

Super Keyword is a reference variable, which is used to invoke immediate parent class constructor. This phenomena is called, Construction chaining in Inheritance.

```
Example: class person {
    int age; String name;
    person(int age, String name)
    {
      this.age = age;
      this.name = name;
    }
}

class emp {
int age, salary;
String name;
emp(int age, String name, int salary)
{ this.salary = salary;
  super(age, name);
}
}

class Test {
public static void main(String args[])
{ emp e1 = new emp(17, 'abc', 30000);
}
}
```

PAGE NO.:
DATE: / /20

## Package:

- Package in Java is a mechanism to encapsulate a group of classes, sub packages and Interfaces. Packages are used for preventing naming conflicts, making searching, locating and uses of classes, Interfaces, easier.
- Package provide controlled access.
- Packages can be considered as Data encapsulation, or data hiding.
- There are 2 types of Packages : * Built in Package
  * User Defined Package

Built in Package: These packages consist of a large number of classes, which are a part of Java API, some commonly used packages are as follows:

1) Java.lang: java.lang supports primitive data types and mathematical operations generally language supported classes are contained in this subpackage. It is imported automatically.

2) Java.io: java.io contains the classes that support Input/Output operations.

3) Java.util: java.util contains utility classes which implement data structures liked linked list, dictionary, date & time operations.

4) Java.awt: java.awt contains classes for Implementing the

components for (Abstract Window Toolkit) Graphical User Interface (GUI) such as Textfield, Radio buttons, etc.

Use of Package: 1) import java.util.*
(* = Clean closure, includes all)

2) import java.p1.first class
(package)

User defined package: * User defined packages are created by the programmer.

Ques)* Write a program in Java to create a user defined package and use the method of the package in another class.

Sol^n
```
package p1;

public class A{
 void m()
 { System.out.println("Hello");
 }
}
```

1) Compile: javac -d . A.java
(. = current directory, -d = destination)

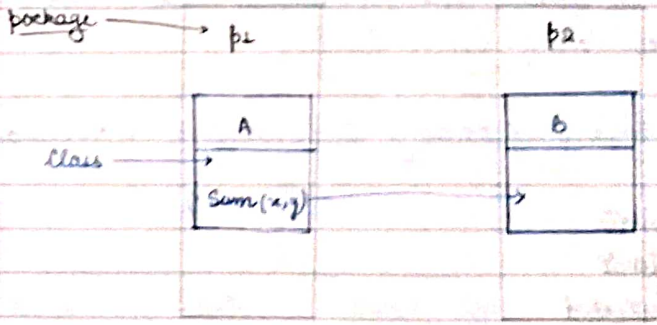
javac -d c:\ A.java

```
import p1.*;
class B
{ public static void main(String args[])
 { A obj = new A();
  obj.m();
 }
}
```

2) javac B.java

3) java B

Ques) Write a program in Java to create a user defined package and use the method of package in another package.




```
package p1;
public class A{
int sum(int x, int y)
 { return x+y;
 }
}
```

Compile:
javac -d . A.java

```
import p1.*;
package p2;

class B {
public static void main(String args[])
{ A obj = new A();
  System.out.println(obj.sum(10,20));
}
}
```

Compile: javac -d . B.java

Run: java p2.B

Access Modifiers: Access modifiers provide the access control to the fields of classes, methods of classes, and to the class itself.

In Java programming, there are four access modifiers

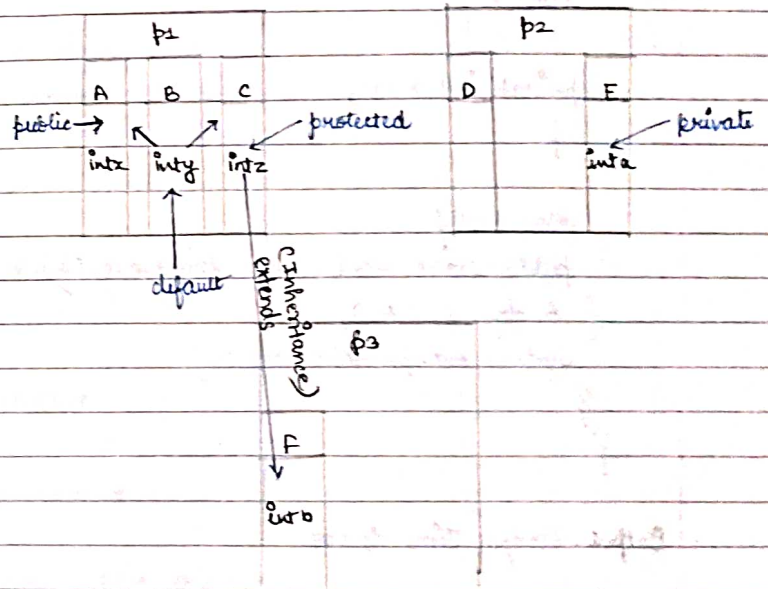
1) private
2) default
3) protected
4) public

* Private access modifiers are the most restricted ones, if any member is private, it means it is accessible, only within a class

* Default members are accessible within a package only.

* Protected access modifier are accessible outside the package, with one condition, i.e. through Inheritance only

* Public access modifiers are the least restricted access modifiers. It means public members are accessible to everywhere.




| Access Modifier | Within class | Outside class within package | Outside package with Inheritance |
|---|---|---|---|
| Private | ✓ | X | X |
| Default | ✓ | ✓ | X |
| Protected | ✓ | ✓ | ✓ |
| Public | ✓ | ✓ | ✓ |

| | outside package (everywhere) |
|---|---|
| Private | X |
| Default | X |
| Protected | X |
| Public | ✓ |

Private:

```
class A {
private int x=10;
}

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

Output: Compile Time Error.

Default:

```
package p1;

class A {
int x=10;
}

package p2;
import p1.*;

class B {

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

Output: Compile Time Error.

Protected:

```
package p1;

public class A {

protected int x=10;
}

package p2;

import p1.*
```

```
class B extends A {

public static void main (String args [])
{
   B obj = new B();
   System.out.println(obj.x);
}
}
```

Output: 10

Encapsulation: * Binding data members and methods in a single unit is called Encapsulation.

* Encapsulation provides Data Hiding.

* Encapsulation provides security to the members of a class in Java programming.

* Encapsulation provides security to the members of a class using private access modifier.

* Encapsulation can make any read only memory or write only memory or both, by utilising setter and getter methods

Example:
```
class Account {
   private int bal = 10000;
}
```

```
class Test {
 public static void main (String args [])
{
 Account a = new Account();
 System.out.println(a.bal);
 }
}
```

* Encapsulation is different from Abstraction because in abstraction implementation details of a method is hidden from the user, whereas in Encapsulation, data members are hidden using private access modifiers (Important)

```
class Account {
private int bal = 10000;
public void set (int a)    // Setter Method
{ bal = bal + a;
 }
public int get ()          // Getter Method.
{ return bal;
 }
}
```

```
class Test {
public static void main (String args [])
{ Account obj = new Account();
 System.out.println(obj.get());
 obj.set(10000);
 System.out.println(obj.get());
 }
}
```

Output:
10000
20000

Ques) Write a program in Java to implement Encapsulation, with two data members.

```
class Addition {
 private int a=10;
 private int b=20;
 public void set(int a1, int b1)
 { a= a+a1;
   b= b+b1;
 }
 public int get1()
 {//return a1;=
  //return b1;=
return a;}

 public int get2()
 {//return a;
   return b;
 }

class Test {
public static void main(String args[])
{ Addition obj = new Addition();
  System.out.println(obj.get1());
  System.out.println(obj.get2());
  obj.set(10,20);
  System.out.println(obj.get1());
  System.out.println(obj.get2());
}
}
```

Output:

10
20
20
40



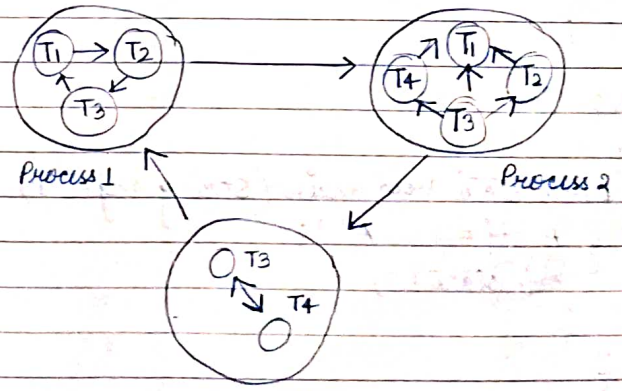
Thread: (Process)

★ Multi Programming / Multi Tasking / User Interaction (Generic Term)
Technique: Multi Threading.

→ A program is set of instruction any program in execution is called Process.

→ When multiple program are executed at a time, then this phenomenon is called multiprogramming. Multiprogramming is achieved by context switching or CPU switching in which CPU is allocated to different processes based on CPU scheduling algorithms such as ★FCFS (First come, First serve) ★ SJF (Shortest Job First) ★ SRTF (Shortest running time first) ★ Roundrobin ★ Priority

→ A light weight process is called thread. Threads are independent to each other, it means 1 thread cannot interfere the execution.



* When a process start executing its thread implicitly start executing at the same time, and the phenomena is called Multithreading.

* There are two types of ways; to implement a thread in Java Programming.
1) by extending a thread class
2) by implementing runnable interface

* Two important methods used in multithreading are,
1) public void run()
2) public void start()

Run method is used to define the working of any thread whereas start method is used to execute the run method.

```
class A extends Thread
{
  void run()
  { System.out.println("Hello");
  }

class Test
{
 public static void main (String args[])
  { A obj = new A();
    obj.start();
  }
}
```

→ Single task multiple threads:

```
class A extends Thread {
void run()
{ for(i=0; i<=1000; i++) {
  System.out.println(i);
 }
}
}

class Test {
public static void main(String args[])
 { A T1= new A();
   A T2 = new A();
   T1.start();
   T2.start();
 }
}
```

Output:
```
0
1
2
0
1
3
4
5
6
2
3
```