

# INHERITANCE

NAME : N SAPTHA NAGESWAR  
REG NO : 192372001  
COURSE : CSA0987- JAVA PROGRAMMING

## Sample Code

Below given is an example demonstrating Java inheritance. In this example you can observe two classes namely Calculation and My\_Calculation. Using extends keyword the My\_Calculation inherits the methods addition and Subtraction of Calculation class. Copy and paste the program given below in a file with name My\_Calculation.java

```
class Calculation{  
    int z;  
    public void addition(int x, int y){  
        z=x+y;  
        System.out.println("The sum of the given numbers:"+z);  
    }  
    public void Substraction(int x,int y){  
        z=x-y;  
        System.out.println("The difference between the given numbers:"+z);  
    }  
}  
  
public class My_Calculation extends Calculation{  
    public void multiplication(int x, int y){  
        z=x*y;  
        System.out.println("The product of the given numbers:"+z);  
    }  
}
```

```

public static void main(String args[]){

int a=20, b=10;

My_Calculation demo = new My_Calculation();

demo.addition(a, b);

demo.Substraction(a, b);

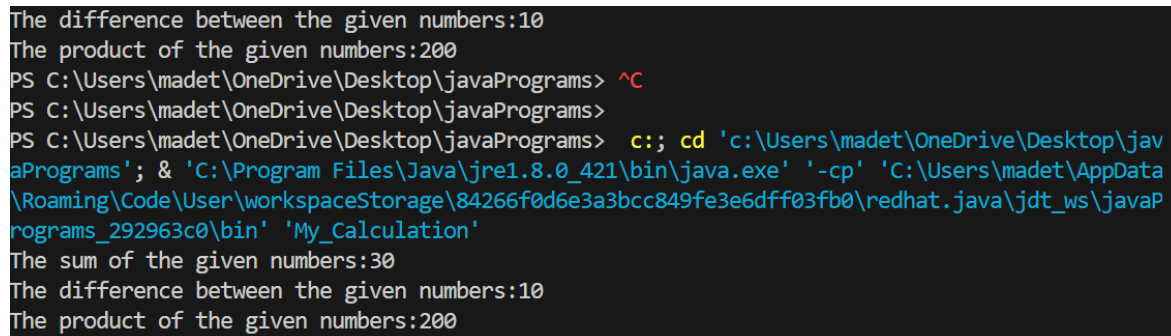
demo.multiplication(a, b);

}

}

```

Output:



```

The difference between the given numbers:10
The product of the given numbers:200
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> c::; cd 'c:\Users\madet\OneDrive\jav
aPrograms'; & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe' '-cp' 'C:\Users\madet\AppData
\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\javaP
rograms_292963c0\bin' 'My_Calculation'
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200

```

## The super keyword

The super keyword is similar to this keyword following are the scenarios where the super keyword is used. It is used to differentiate the members of superclass from the members of subclass, if they have same names. It is used to invoke the superclass constructor from subclass. Differentiating the members If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below. super.variable super.method();

## Sample Code

This section provides you a program that demonstrates the usage of the super keyword. In the given program you have two classes namely Sub\_class and Super\_class, both have a method named display with different implementations, and a variable named num with different values. We are invoking display method of both classes and printing the value of the variable num of both classes, here you can observe that we have used super key word to differentiate the members of super class from sub class.

```

class Super_class{

```

```

int num=20;

//display method of superclass
public void display(){
System.out.println("This is the display method of superclass");
}
}

public class Sub_class extends Super_class {
int num=10;

//display method of sub class
public void display(){
System.out.println("This is the display method of subclass");
}

public void my_method(){
//Instantiating subclass
Sub_class sub=new Sub_class();
//Invoking the display() method of sub class
sub.display();

//Invoking the display() method of superclass
super.display();

//printing the value of variable num of subclass
System.out.println("value of the variable named num in sub class:"+ sub.num);

//printing the value of variable num of superclass
System.out.println("value of the variable named num in super class:"+ super.num);
}

public static void main(String args[]){
Sub_class obj = new Sub_class();
obj.my_method();
}
}

```

```

PS C:\Users\madet\OneDrive\Desktop\javaPrograms> & 'C:\Program Files\Java\jre1.8.0_421\bin\j
ava.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58568' '-cp
' 'C:\Users\madet\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0
\redhat.java\jdt_ws\javaPrograms_292963c0\bin' 'Sub_class'
This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>

```

## Invoking Superclass constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class. But if you want to call a parametrized constructor of the super class, you need to use the super keyword as shown below. `super(values);`

## Sample Code

The program given in this section demonstrates how to use the super keyword to invoke the parametrized constructor of the superclass. This program contains a super class and a sub class, where the super class contains a parametrized constructor which accepts a string value, and we used the super keyword to invoke the parametrized constructor of the super class.

```

class Superclass{

    int age;

    Superclass(int age){

        this.age=age;

    }

    public void getAge(){

        System.out.println("The value of the variable named age in super class is: " +age);

    }

}

public class Subclass extends Superclass {

    Subclass(int age){

        super(age);

    }

    public static void main(String argd[]){

        Subclass s= new Subclass(24);

        s.getAge();

    }

}

```

```
}
```

```
3c0\bin' 'Subclass'  
The value of the variable named age in super class is: 24  
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

## Example

```
class Animal{  
  
}  
  
class Mammal extends Animal{  
  
}  
  
class Reptile extends Animal{  
  
}  
  
public class Dog extends Mammal{  
    public static void main(String args[]){  
        Animal a = new Animal();  
        Mammal m = new Mammal();  
        Dog d = new Dog();  
        System.out.println(m instanceof Animal);  
        System.out.println(d instanceof Mammal);  
        System.out.println(d instanceof Animal);  
    }  
}
```

}

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> c::; cd 'c:\Users\madet\OneDrive\Desktop\javaPrograms'; & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58764' '-cp' 'C:\Users\madet\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\javaPrograms_292963c0\bin' 'Dog'
true
true
true
```

# POLYMORPHISM

## Virtual Methods:

In this section, I will show you how the behaviour of overridden methods in Java allows you to take advantage of polymorphism when designing your classes. We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

```
public class Employee
{
    private String name;
    private String address;
    private int number;

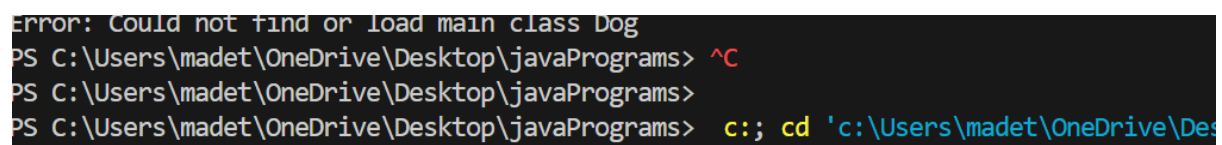
    public Employee(String name, String address, int
    number)
    {
        System.out.println("Constructing an Employee");
        this.name = name;
```

```
this.address = address;
this.number = number;
}
public void mailCheck()
{
    System.out.println("Mailing a check to " + this.name
+ " " + this.address);
}
public String toString()
{
    return name + " " + address + " " + number;
}
public String getName()
{
    return name;
}
public String getAddress()
{
    return address;
}
```



```
public void setAddress(String newAddress)
{
    address = newAddress;
}

public int getNumber()
{
    return number;
}
}
```



```
error: Could not find or load main class Dog
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> c:; cd 'c:\Users\madet\OneDrive\Des
```

Now suppose we extend Employee class as follows:

```
public class Salary extends Employee
{
    private double salary; //Annual salary

    public Salary(String name, String address, int number,
        double
        salary)
    {
        super(name, address, number);
        setSalary(salary);
    }
}
```

```
}  
  
public void mailCheck()  
{  
    System.out.println("Within mailCheck of Salary class ");  
    System.out.println("Mailing check to " + getName()  
+ " with salary " + salary);  
}  
  
public double getSalary()  
{  
    return salary;  
}  
  
public void setSalary(double newSalary)  
{  
    if(newSalary >= 0.0)  
    {  
        salary = newSalary;  
    }  
}  
  
public double computePay()  
{
```

```
System.out.println("Computing salary pay for " +  
getName());  
return salary/52;  
  
}  
  
}
```

```
Constructing an Employee  
Constructing an Employee  
Call mailCheck using Salary reference --  
Within mailCheck of Salary class  
mailing check to Mohd Mohtashim with salary 3600.0  
  
Call mailCheck using Employee reference--  
Within mailCheck of Salary class  
mailing check to John Adams with salary 2400.0
```

## MULTI THREADING

Java is a multi threaded programming language which means we can develop multi threaded program using Java. A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Above-mentioned stages are explained here:

New:

A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

Waiting: Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

Timed waiting: A runnable thread can enter the timed waiting state for a specified interval of time. A thread in

this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

**Terminated Dead:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## **Create Thread by Implementing Runnable Interface:**

### Step 1:

As a first step you need to implement a run method provided by Runnable interface. This method provides entry point for the thread and you will put your complete business logic inside this method. Following is simple syntax of run method:

```
public void run( )
```

### Step 2:

At second step you will instantiate a Thread object using the following constructor:

```
Thread(Runnable threadObj, String threadName);
```

### Step 3

Once Thread object is created, you can start it by calling start method, which executes a call to run method. Following is simple syntax of start method:

```
void start( );
```

```
class RunnableDemo implements Runnable {  
    private Thread t;  
    private String threadName;  
    RunnableDemo( String name){  
        threadName = name;  
        System.out.println("Creating " + threadName );  
    }  
    public void run() {  
        System.out.println("Running " + threadName );  
        try {  
            for(int i = 4; i > 0; i--) {  
                System.out.println("Thread: " + threadName + ", " +  
i);  
                // Let the thread sleep for a while.  
                Thread.sleep(50);  
            }  
        }  
    }  
}
```

```
    } catch (InterruptedException e) {  
        System.out.println("Thread " + threadName + "  
interrupted.");  
    }  
    System.out.println("Thread " + threadName + "  
exiting.");  
}  
public void start ()  
{  
    System.out.println("Starting " + threadName );  
    if (t == null)  
    {  
        t = new Thread (this, threadName);  
        t.start ();  
    }  
}  
}  
  
public class TestThread {  
    public static void main(String args[]) {  
        RunnableDemo R1 = new RunnableDemo( "Thread-  
1");
```

```
R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-
2");

R2.start();

}

}
```

```
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe'
'-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:59275' '-cp' 'C:\Users\ma
det\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\j
avaPrograms_292963c0\bin' 'TestThread'
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-2, 3
Thread: Thread-1, 3
Thread: Thread-2, 2
Thread: Thread-1, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
```

## Create Thread by Extending Thread Class:

The second way to create a thread is to create a new class that extends Thread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.



## Step 1

You will need to override run method available in Thread class. This method provides entry point for the thread and you will put your complete business logic inside this method. Following is simple syntax of run method:

```
public void run( )
```

Example:

Here is the preceding program rewritten to extend Thread

```
class ThreadDemo extends Thread {  
    private Thread t;  
    private String threadName;  
    ThreadDemo( String name){  
        threadName = name;  
        System.out.println("Creating " + threadName );  
    }  
    public void run() {  
        System.out.println("Running " + threadName );  
        try {  
            for(int i = 4; i > 0; i--) {
```

```
        System.out.println("Thread: " + threadName + ", " +  
i);  
        // Let the thread sleep for a while.  
        Thread.sleep(50);  
    }  
    } catch (InterruptedException e) {  
        System.out.println("Thread " + threadName + "  
interrupted.");  
    }  
    System.out.println("Thread " + threadName + "  
exiting.");  
    }  
    public void start ()  
    {  
        System.out.println("Starting " + threadName );  
        if (t == null)  
        {  
            t = new Thread (this, threadName);  
            t.start ();  
        }  
    }  
}
```

```

}

public class TestThread {

    public static void main(String args[]) {

        ThreadDemo T1 = new ThreadDemo( "Thread-1");

        T1.start();

        ThreadDemo T2 = new ThreadDemo( "Thread-2");

        T2.start();

    }

}

```

```

File Edit Selection View Go ...
javaPrograms

EXPLORER
JAVAPROGRAMS
TestThread.java 4

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> ^C
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>
PS C:\Users\madet\OneDrive\Desktop\javaPrograms> c:; cd 'c:\Users\madet\OneDrive\Desktop\javaPrograms'; & 'C:\Program Files\Java\jre1.8.0_421\bin\java.exe' -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:59396' '-cp' 'C:\Users\madet\AppData\Roaming\Code\User\workspaceStorage\84266f0d6e3a3bcc849fe3e6dff03fb0\redhat.java\jdt_ws\javaPrograms_292963c0\bin' 'TestThread'
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-2, 2
Thread: Thread-1, 2
Thread: Thread-2, 1
Thread: Thread-1, 1
Thread Thread-2 exiting.
Thread Thread-1 exiting.
PS C:\Users\madet\OneDrive\Desktop\javaPrograms>

Run: Armstr...
Run: Sub_d...
Run: My_Cal...
Debug: Sub...
Debug: Sub...
Debug: Dog
Run: Dog
Run: TestThr...
Debug: Test...

```