Inheritance:
==========

--Getting the properties of one object of a class to another Object of another class is known as inhetance.

--with the help of inheritance we achieve IS-A realtionship

--the main adv of inheritance is code reusability and run time polymorphism.

Has-A relationship example:

```
class A {

B b1 = new B();


}
```

```
Animal // parent/super/base

sleep() //1000
walk() //5000
eat() //6000
makeNoise()


//child/sub/derived
Dog    Cat     Tiger   Lion ,20
```

Type of inheritance:
==========

Note: At class level java supports only multilevel inheritance(single, hirarical).where as with the help of interface we can achieve multiple inheritance.

A.java:

```
---------

package com.masai;

public class A {

	int i=10;

	void funA() {

		System.out.println("inside funA of A");
	}

}
```

Demo.java:

```
--------------

package com.masai;

public class Demo extends A{

	int x=100;

	void funX() {

		System.out.println("inside funX of Demo");
	}


	public static void main(String[] args) {

		Demo d1 = new Demo();

		System.out.println(d1.x);
		d1.funX();

		System.out.println(d1.i);
```

```
        d1.funA();

    }

}
```

Here A class will become parent/super class and Demo class will become child clas or subclass.

--the the object of child class we can reffer the properties of parent class also.

With respect to the diagram:

1. Object of the super class is created first completly before object of the child class created completly.

2.super class obj will be created from the constructor of the subclass.(by using super();)

```
Demo(){
super(); // create the parent class object by executing parent class constructor.

}
```

3. super class object will be created in association with the subclass object.

4. Object class is the super most class of any java class.

Note: meaning of super() is to create the parent class object by executing parent class zero arguement constructor.

Note: if inside the parent class we have only parameterized constructor then inside the child class we need to call that parent class constructor explicitly by supplying appropriate argument ex: super(10);
otherwise our inheritance will fail.. here we can not depend upon default constructor of the child class, we need to keep constructor inside the child class explicitly and from there need to call parent class constrcutor,

example:

A.java:

```
package com.masai;

public class A {

	int i=10;

	A(int i){
		System.out.println("inside A constructor");

	}

	void funA() {

		System.out.println("inside funA of A");
	}

}
```

Demo.java:

```
package com.masai;

public class Demo extends A{

	int x=100;


	Demo(){
		super(10);
		System.out.println("inside Demo constructor");
	}

	void funX() {

		System.out.println("inside funX of Demo");
	}


	public static void main(String[] args) {
```

```
		Demo d1 = new Demo();

		System.out.println(d1.x);
		d1.funX();

		System.out.println(d1.i);
		d1.funA();

		d1.toString();

	}

}
```

Method overriding:--
==========

--it is the procedure of defining a method in the sub-class with the same name and same signature which is already defined inside the parent class/super class.

A.java:

```
package com.masai;

public class A {

	int i=10;


	void funA() {

		System.out.println("inside funA of A");
	}

}
```

Demo.java:

```
package com.masai;

public class Demo extends A{

	int x=100;



	void funX() {

		System.out.println("inside funX of Demo");
	}

	@Override //this annotation make sure that we have overide correct method.
	void funA() {

		System.out.println("inside funA of Demo");
		System.out.println("This is second statement inside funA of Demo");
	}


	public static void main(String[] args) {

		Demo d1 = new Demo();

		System.out.println(d1.x);
		d1.funX();

		System.out.println(d1.i);
		d1.funA(); // overriden method will gets the priority
	}

}
```

method overriding rules:
----

1. while overriding a method inside the child class we can not reduce its accessebility. but we can increase.

2. related with the execption handling.

super keyword:

--super keyword will represent the immediate parent class object.
--this keyword represents the current class object.

--there are 2 uses of the super keyword:

1.to represent the immediate parent class object.

2.to call the parent class constructor.

Note: by super keyword calling the method of parent class and calling the constructor of the parent class is entirly different.
--constructor call must be the first statement while method call can be any place.

******Note: we can not use super and this keyword inside the static area.**

example:

A.java:

```
package com.masai;

public class A {

	int i=10;


	void funA() {

		System.out.println("inside funA of A");
	}



}
```

Demo.java:

```
package com.masai;

public class Demo extends A{

	int x=100;


	@Override
	void funA() {

		System.out.println("inside funA of Demo");
		System.out.println("This is second statement inside funA of Demo");
	}


	void funX() {

		System.out.println("inside funX of Demo");
		funA(); //overriden method will be called
		super.funA(); //parent class method will be called



	}



	public static void main(String[] args) {

		Demo d1 = new Demo();

		d1.funX();
		//super.d1.funA() // ERROR
```

```
	}

}
```

*****Note: static members will not participate in inheritance.even though through the child class object we can access the static members of the parent class. these members will be accessed from the context area. not from the object.

--if we define same static method inside the child class which is already defined statically inside the parent class, then this concept is known as method hinding, it is not a method overriding, it will hide the parent class static method inside the context area.

example:

A.java:

```
package com.masai;

public class A {

	int i=10;


	static void funS() {

		System.out.println("inside funS of A");
	}



	void funA() {

		System.out.println("inside funA of A");
	}



}
```

Demo.java:

```
package com.masai;

public class Demo extends A{

	int x=100;


	@Override
	void funA() {

		System.out.println("inside funA of Demo");
		System.out.println("This is second statement inside funA of Demo");
	}


	void funX() {

		System.out.println("inside funX of Demo");
		funA(); //overriden method will be called
		super.funA(); //parent class method will be called



	}


	static void funS() {

		System.out.println("inside funS of Demo");
	}




	public static void main(String[] args) {

		Demo d1 = new Demo();

		d1.funX();

		d1.funS();// we can access the static with the child class object also.
		A.funS();
```

```
            Demo.funS();


        }

}
```

Super class reference and child class object:
==========

--according to this rule, we can assign any child class object to its parent class reference variable.

```
A a1 = new A();
A a1 = null;
A a1 = new Demo(); // this is possible only if Demo class is the child class of A
```

A.java:

```
funA()
funB()
```

Demo.java extends A

```
funA(); //overrident
funX(); // child specific method
```

Note: if a super class ref pointing to the child class object, then with that obj we can refer only members of the parent class and, if that method is overriden inside the child class then child class method will gets the priority.
but with the super class reference we can not call the child class specific functionality. will get a CE.

example:

A.java:

```
package com.masai;

public class A {

	int i=10;

	void funA() {

		System.out.println("inside funA of A");
	}

	void funB() {
		System.out.println("inside funB of A");
	}


}
```

Demo.java:

```
package com.masai;

public class Demo extends A{

	int x=100;


	@Override
	void funA() {

		System.out.println("inside funA of Demo");
	}


	void funX() {

		System.out.println("inside funX of Demo");
	}
```

```
	public static void main(String[] args) {

//		Demo d1 = new Demo();
//
//		d1.funB();
//		d1.funA();//Demo
//		d1.funX();


		//A a1 = new A();

		//a1.funA(); // A
		//a1.funX(); //CE


		A a1 = new Demo();

		a1.funA();
		a1.funB();

		//a1.funX(); //CE




	}

}
```

Note: in order to call the child class specific functionality from the parent class object, we need to downcast the super class ref to the appropriate child class object.

ex:

```
	//upcasting
	A a1= new Demo();

	//downcasting
	Demo d1= (Demo)a1;

	d1.funX();
```

//object downcasting this is possible only when super class ref points to the child class object
//if A a1 = new A() and then we try to downcast then we will get a runtime exception called ClassCastException