

Inheritance in Java

NAME: S. Ajay Kumar

REGISTER NUMBER: 2024503701

1.1 CODE:

```
class LivingBeing{
    void breath(){
        System.out.println("LivingBeings can Breath...");
    }
    void response(){
        System.out.println("LivingBeings can Respond...");
    }
}

class Animal extends LivingBeing{
    void walk(){
        System.out.println("Animals can Walk...");
    }
    void noOfLegs(){
        System.out.println("Animals can walk on 2 or 4 legs...");
    }
}

public class Single{
    public static void main(String[] args) {
        System.out.println("Name : S.Ajay Kumar");
        System.out.println("Roll no : 2024503701");
    }
}
```

```
Animal animal = new Animal();  
animal.breath();  
animal.response();  
animal.walk();  
animal.noOfLegs();  
}  
}
```

OUTPUT:

```
Name : S.Ajay Kumar  
Roll no : 2024503701  
LivingBeings can Breath...  
LivingBeings can Respond...  
Animals can Walk...  
Animals can walk on 2 or 4 legs...
```

1.2 CODE:

```
class LivingBeing{  
    void breath(){  
        System.out.println("LivingBeings can Breath...");  
    }  
    void response(){  
        System.out.println("LivingBeings can Respond...");  
    }  
}  
  
class Animal extends LivingBeing{  
    void walk(){  
        System.out.println("Animals can Walk...");  
    }  
}
```

```
void noOfLegs() {
    System.out.println("Animals can walk on 2 or 4 legs...");
}
}

class Cat extends Animal {
    void meow() {
        System.out.println("Cats can Meow...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dogs can Bark...");
    }
}

public class Multilevel {
    public static void main(String[] args) {
        System.out.println("Name : S.Ajay Kumar");
        System.out.println("Roll no : 2024503701");
        Dog dog = new Dog();
        Cat cat = new Cat();
        dog.breath();
        dog.response();
        dog.walk();
        dog.noOfLegs();
        dog.bark();
        cat.meow();
    }
}
```

```
}  
}
```

OUTPUT:

```
Name : S.Ajay Kumar  
Roll no : 2024503701  
LivingBeings can Breath...  
LivingBeings can Respond...  
Animals can Walk...  
Animals can walk on 2 or 4 legs...  
Dogs can Bark...  
Cats can Meow...
```

1.3 CODE:

```
class Animals{  
    void move(){  
        System.out.println("Animals can move...");  
    }  
    void move(String direction){  
        System.out.println("Animals can move " + direction + "  
direction...");  
    }  
    void move(int distance){  
        System.out.println("Animals can move " + distance + " meters...");  
    }  
    void move(String direction, int distance){  
        System.out.println("Animals can move " + direction + " for " +  
distance + " meters...");  
    }  
}
```

```

    }
}

class Dogs extends Animals{
    @Override
    void move(){
        System.out.println("Dogs can run...");
    }
    @Override
    void move(String direction){
        System.out.println("Dogs can run in " + direction + " direction...");
    }
    @Override
    void move(int distance){
        System.out.println("Dogs can run " + distance + " meters...");
    }
    @Override
    void move(String direction, int distance){
        System.out.println("Dogs can run " + direction + " for " + distance +
" meters...");
    }
}

class Cats extends Animals{
    @Override
    void move(){
        System.out.println("Cats can jump...");
    }
    @Override

```

```

void move(int distance){
    System.out.println("Cats can jump " + distance + " meters...");
}
@Override
void move(String direction){
    System.out.println("Cats can jump in " + direction + " direction...");
}
@Override
void move(String direction , int distance){
    System.out.println("Cats can jump " + direction + " for " + distance
+ " meters...");
}
}

class Bird extends Animals{
    @Override
    void move(){
        System.out.println("Birds can fly");
    }
    @Override
    void move(String direction){
        System.out.println("Birds can fly in " + direction + " direction...");
    }
    @Override
    void move(int distance){
        System.out.println("Birds can fly " + distance + " meters...");
    }
    @Override

```

```
void move(String direction , int distance){  
    System.out.println("Birds can fly " + direction + " for " + distance +  
    " meters...");  
}  
  
public class Main3 {  
    public static void main(String[] args) {  
        System.out.println("Name : S.Ajay Kumar");  
        System.out.println("Roll no : 2024503701");  
        Animals[] animalArray;  
        animalArray = new Animals[3];  
        animalArray[0] = new Dogs();  
        animalArray[1] = new Cats();  
        animalArray[2] = new Bird();  
  
        for (Animals animal : animalArray) {  
            animal.move();  
            animal.move("North");  
            animal.move(100);  
            animal.move("East", 50);  
            System.out.println();  
        }  
  
        Dogs dog = new Dogs();  
        dog.move();  
        dog.move("South");  
        dog.move(90);  
    }  
}
```

```
dog.move("East", 99);
```

```
}
```

```
}
```

OUTPUT:

```
Name : S.Ajay Kumar
Roll no : 2024503701
Dogs can run...
Dogs can run in North direction...
Dogs can run 100 meters...
Dogs can run East for 50 meters...

Cats can jump...
Cats can jump in North direction...
Cats can jump 100 meters...
Cats can jump East for 50 meters...

Birds can fly
Birds can fly in North direction...
Birds can fly 100 meters...
Birds can fly East for 50 meters...

Dogs can run...
Dogs can run in South direction...
Dogs can run 90 meters...
Dogs can run East for 99 meters...
```

1.4 CODE:

```
class LivingBeings{
```



```
public LivingBeings(String name) {  
    System.out.println("Name : " + name);  
}  
void breath() {  
    System.out.println("LivingBeings can Breath...");  
}  
void response() {  
    System.out.println("LivingBeings can respond...");  
}  
  
}  
class LandAnimal extends LivingBeings {  
    public LandAnimal(String name) {  
        super(name);  
    }  
    void walk() {  
        System.out.println("LandAnimal can walk...");  
    }  
    void noOfLegs() {  
        System.out.println("LandAnimals can walk on 2 or 4 legs...");  
    }  
}  
  
class Dog extends LandAnimal {  
    public Dog(String name) {  
        super(name);  
    }  
    void bark() {
```

```
        System.out.println("Dog can bark...");
    }
}

class Cat extends LandAnimal {
    public Cat(String name) {
        super(name);
    }
    void meow() {
        System.out.println("Cat can meow...");
    }
}

class Main4 {
    public static void main(String[] args) {
        System.out.println("Name : S.Ajay Kumar");
        System.out.println("Roll no : 2024503701");
        Dog dog = new Dog("Tommy");
        Cat cat = new Cat("Kitty");
        dog.breath();
        dog.response();
        cat.breath();
        cat.response();
    }
}
```

OUTPUT:

```
Name : S.Ajay Kumar  
Roll no : 2024503701  
Name : Tommy  
Name : Kitty  
LivingBeings can Breath...  
LivingBeings can respond...  
LivingBeings can Breath...  
LivingBeings can respond...
```

1.5.1 CODE:

```
class Parent {  
    public int show() {  
        return 10;  
    }  
}  
  
class Child extends Parent {  
  
    @Override  
    public String show() {  
        return "Hello";  
    }  
}  
  
public class five1 {  
    public static void main(String[] args) {  
        Parent obj = new Child();  
        System.out.println(obj.show());  
    }  
}
```

```
}
```

OUTPUT:

J five1.java 1

💡 The return type is incompatible with Parent.show() Java(67109268) [Ln 10, Col 12]

1.5.2 CODE:

```
class Parent {  
    public void display() {  
        System.out.println("Parent display");  
    }  
}
```

```
class Child extends Parent {  
  
    @Override  
    private void display() {  
        System.out.println("Child display");  
    }  
}
```

```
public class five2 {  
    public static void main(String[] args) {  
        Parent obj = new Child();  
        obj.display();  
    }  
}
```

```
}
```

OUTPUT:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    The method display() is undefined for the type Parent

    at five2.main(five2.java:18)
PS D:\MIT\JAVA\week 6>
```

1.5.3 CODE:

```
class Parent {
    public void show() {
        System.out.println("Parent show()");
    }
}
```

```
class Child extends Parent {

    @Override
    public void show(int x) {
        System.out.println("Child show(int): " + x);
    }
}
```

```
public class five3 {
    public static void main(String[] args) {
        Parent p = new Child();
        p.show();
    }
}
```

OUTPUT:

five3.java 1

💡 The method show(int) of type Child must override or implement a supertype method Java(67109498) [Ln 10, Col 17]

1.5.4 CODE:

```
class Test {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public double add(int a, int b) {  
        return (double)(a + b);  
    }  
}  
  
public class five4 {  
    public static void main(String[] args) {  
        Test t = new Test();  
        System.out.println(t.add(5, 10));  
    }  
}
```

OUTPUT:

five4.java 2

⊗ Duplicate method add(int, int) in type Test Java(67109219) [Ln 2, Col 16]

💡 Duplicate method add(int, int) in type Test Java(67109219) [Ln 7, Col 19]

1.5.5 CODE:

```
class Parent {

    public Object getValue() {
        return "Parent Object";
    }
}

class Child extends Parent {

    @Override
    public String getValue() {
        return "Child String";
    }
}

public class five5 {
    public static void main(String[] args) {
        Parent p = new Parent();
        System.out.println(p.getValue());

        Parent c = new Child();
        System.out.println(c.getValue());
    }
}
```

OUTPUT:

```
Parent Object  
Child String  
PS D:\MIT\JAVA\week 6> █
```

1.5.6 CODE:

```
class Parent {  
    public void show() {  
        System.out.println("Parent show()");  
    }  
  
    public void show(int x) {  
        System.out.println("Parent show(int): " + x);  
    }  
}  
  
class Child extends Parent {  
    public void show(String msg) {  
        System.out.println("Child show(String): " + msg);  
    }  
  
    public void show(int x, int y) {  
        System.out.println("Child show(int, int): " + (x + y));  
    }  
}  
  
public class five6 {
```



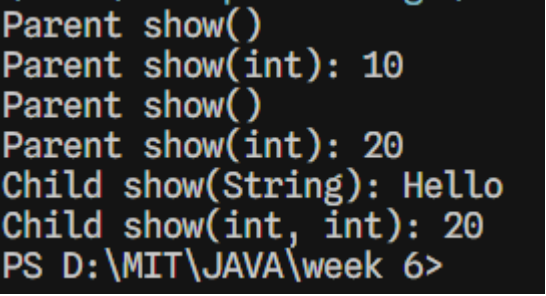
```

    public static void main(String[] args) {
System.out.println("Name : S.Ajay Kumar");
    System.out.println("Roll no : 2024503701");
        Parent p = new Parent();
        p.show();
        p.show(10);

        Child c = new Child();
        c.show();
        c.show(20);
        c.show("Hello");
        c.show(5, 15);
    }
}

```

OUTPUT:



```

Parent show()
Parent show(int): 10
Parent show()
Parent show(int): 20
Child show(String): Hello
Child show(int, int): 20
PS D:\MIT\JAVA\week 6>

```

1.5.7 CODE:

```

class Test {
    public static void display() {
        System.out.println("display()");
    }
}

```

```

public static void display(int x) {
    System.out.println("display(int): " + x);
}

public static void display(String msg) {
    System.out.println("display(String): " + msg);
}

public static void display(int x, int y) {
    System.out.println("display(int, int): " + (x + y));
}
}

public class five7 {
    public static void main(String[] args) {
        Test.display();
        Test.display(10);
        Test.display("Hello");
        Test.display(5, 15);
    }
}

```

OUTPUT:

```

display()
display(int): 10
display(String): Hello
display(int, int): 20
PS D:\MIT\JAVA\week 6>

```

1.5.8 CODE:

```
class Parent {  
    public void show() {  
        System.out.println("Parent show()");  
    }  
}
```

```
class Child extends Parent {  
    public void display() {  
        System.out.println("Child display()");  
    }  
}
```

```
class AnotherChild extends Parent {  
    public void print() {  
        System.out.println("AnotherChild print()");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Parent p1 = new Child();  
        p1.show();  
  
        Child c1 = (Child) p1;  
        c1.show();  
        c1.display();  
    }  
}
```

```

try {
    Parent p2 = new Parent();
    Child c2 = (Child) p2;
    c2.display();
} catch (ClassCastException e) {
    System.out.println("Wrong Downcasting: " + e);
}

```

```

Parent p3 = new AnotherChild();
p3.show();

```

```

try {
    Child c3 = (Child) p3;
    c3.display();
} catch (ClassCastException e) {
    System.out.println("Wrong Downcasting (AnotherChild → Child):"
        + e);
}
}
}

```

OUTPUT:

```

Parent show()
Parent show()
Child display()
Wrong Downcasting: java.lang.ClassCastException: class Parent cannot be cast to class Child (Parent and Child are in unnamed module of loader 'app')
Parent show()
Wrong Downcasting (AnotherChild ? Child): java.lang.ClassCastException: class AnotherChild cannot be cast to class Child (AnotherChild and Child are in unnamed module of loader 'app')
PS D:\MIT\JAVA\week 6>

```