Cambodian University of Specialties

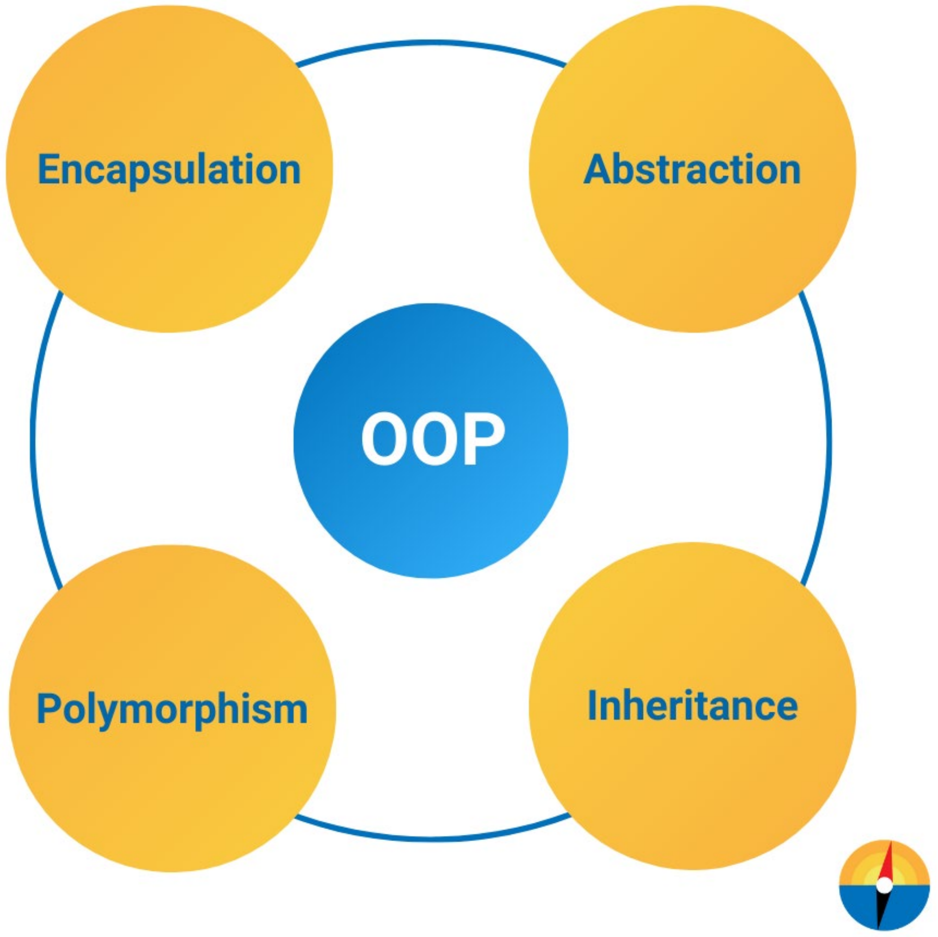
Faculty of Science and Technology

Java OOP I

Introduction to Java Programming

- Introduction to principle OOP
- Inheritance
- Overriding

Introduction to OOP in Java



Introduction to OOP in Java

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which are data structures that contain data, in the form of fields (or attributes) and code, in the form of procedures, (or methods).

Introduction to OOP in Java

- Object-oriented programming has several advantages over procedural programming:
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Introduction to OOP in Java

class	objects
Fruit	Apple
	Banana
	Mango

Introduction to OOP in Java

class	objects
Car	Volvo
	Audi
	Toyota

Introduction to OOP in Java

```
class Car {
        // fields String type;
        String model;
        String color;
 int speed;
}
```

```
// constructor
Car(String type, String model,
String color)
{ this.type = type;
this.model      =      model;
this.color = color;
}
```

Introduction to OOP in Java

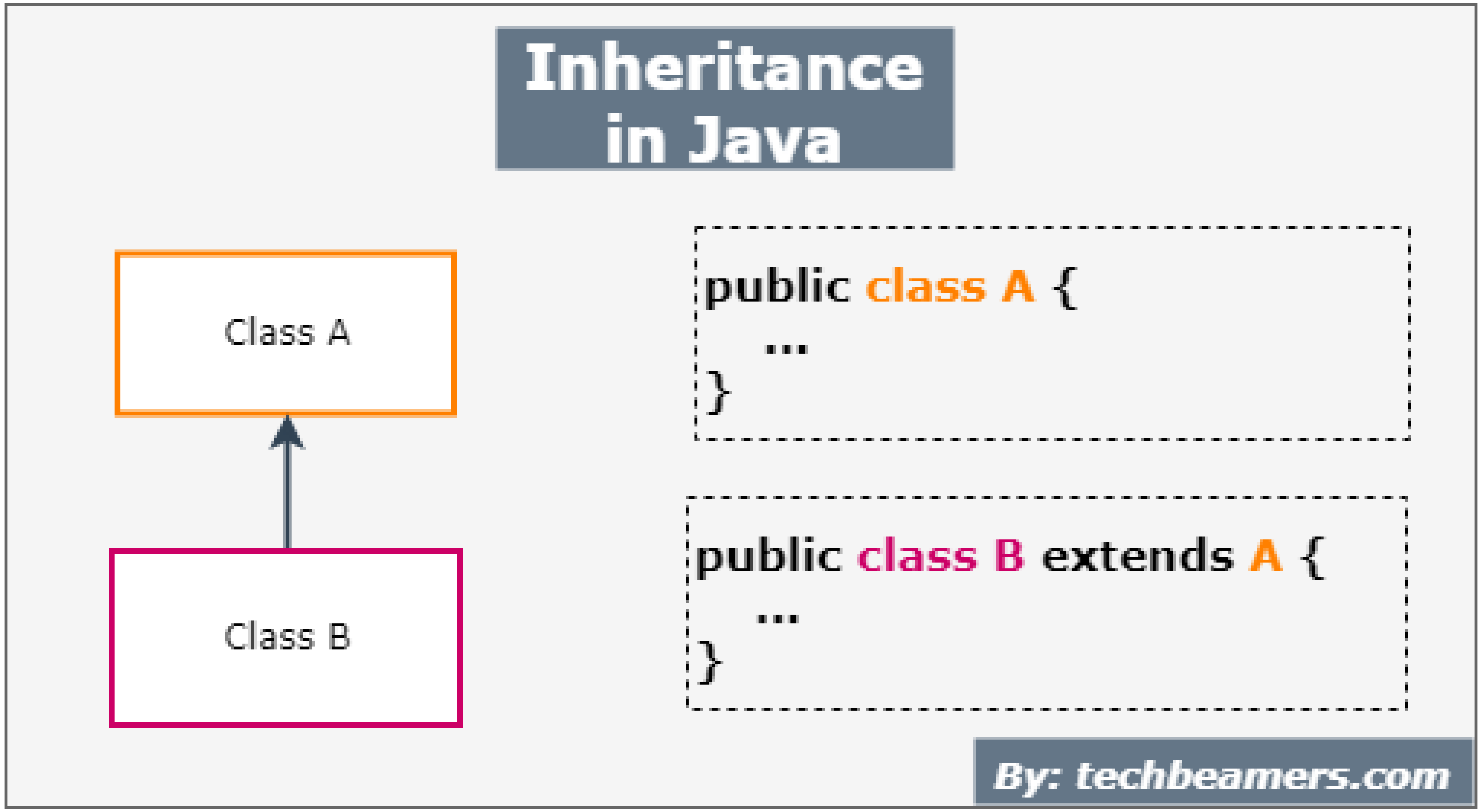
```
// methods
int increaseSpeed(int increment)
{ this.speed = this.speed + increment;
return this.speed;
 }
```

```
Car focus = new Car("Ford", "Focus", "red");
Car auris = new Car("Toyota", "Auris", "blue");
Car golf = new Car("Volkswagen", "Golf", "green");
```

Java OOP Inheritance

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Java OOP Inheritance



```
public class A {
    ...
}
```

```
public class B extends A {
    ...
}
```

Java OOP Inheritance

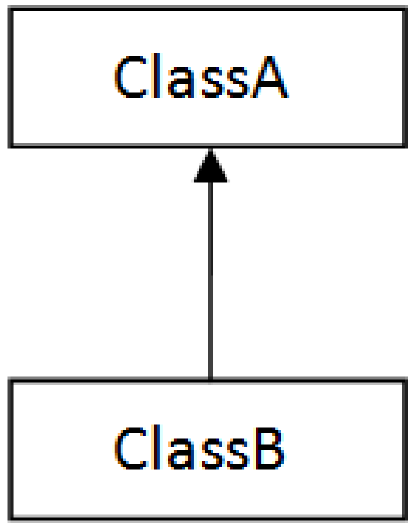
```
class Calculator {
    int add(int a , int b)
    {
        return a + b;
    }

    int sub(int a , int b)
    {
        return a - b;
    }
}
```

```
public class AdvancedCalculator extends Calculator {
    int mult(int a , int b)
    {
        return a * b;
    }

    int div(int a , int b)
    {
        return a / b;
    }
```

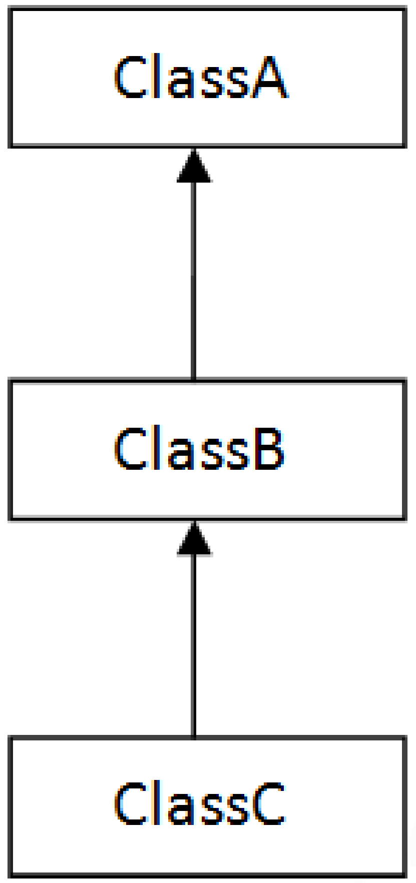
Java OOP Inheritance

```
public static void main(String args[])
{
    AdvancedCalculator cal = new AdvancedCalculator();

    System.out.println( cal.add(1, 2) );
    System.out.println( cal.sub(1, 2) );
    System.out.println( cal.mult(1, 2) );
    System.out.println( cal.div(1, 2) );
}
```

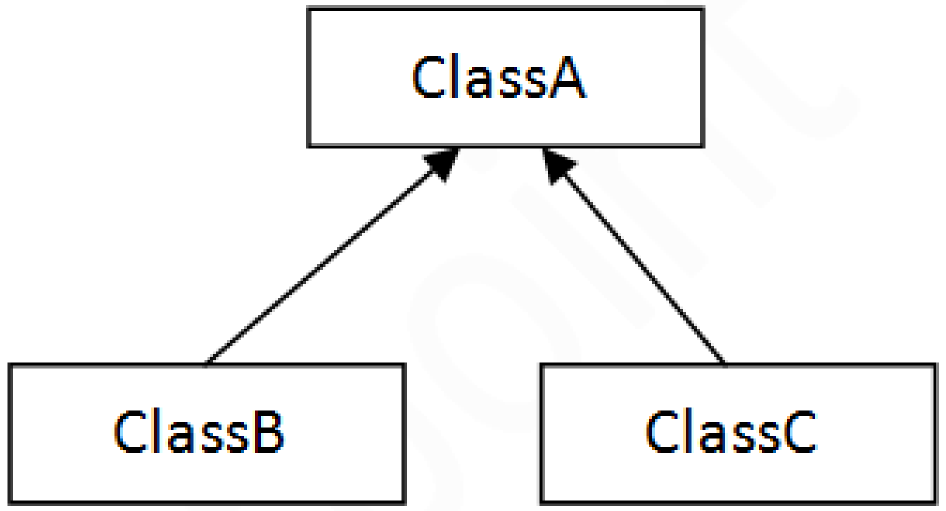
Types of inheritance in java



1) Single

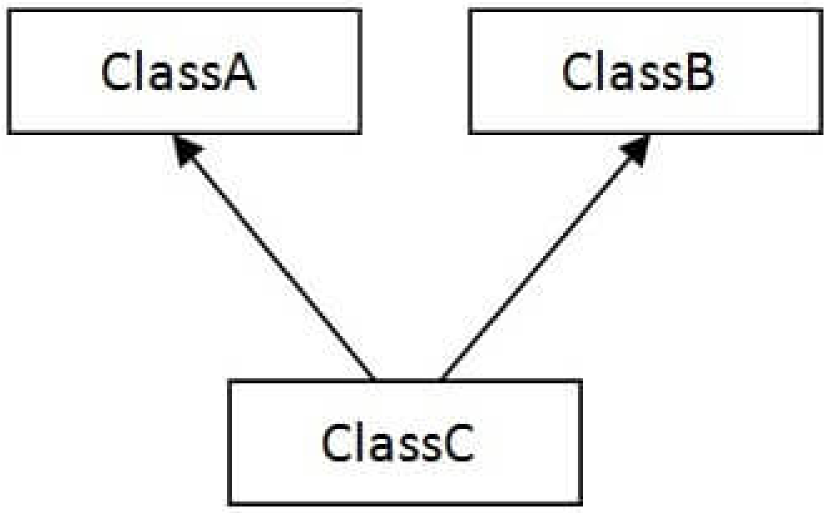


2) Multilevel

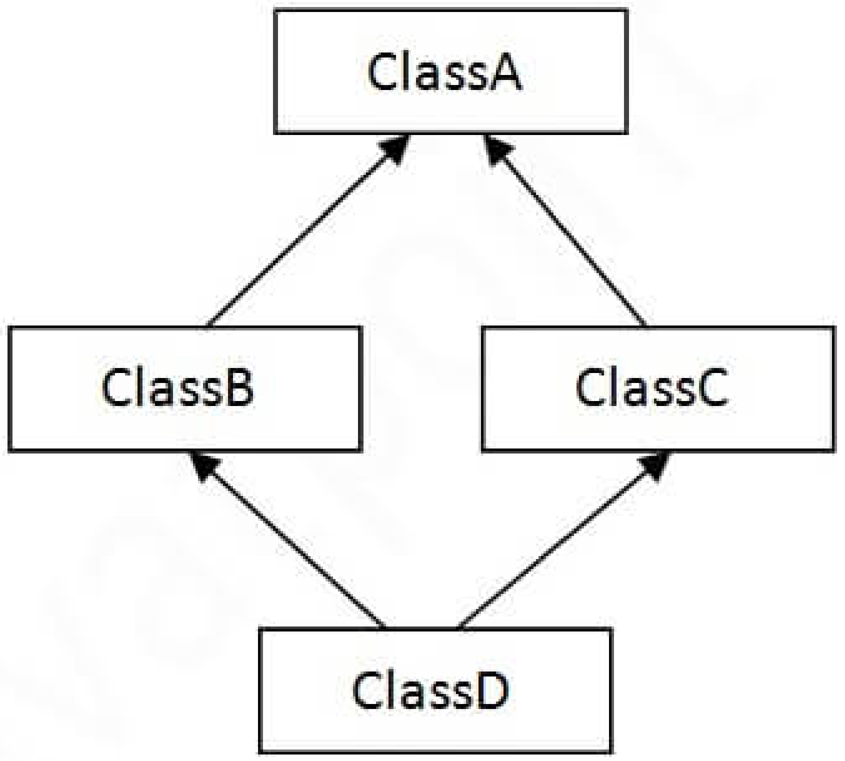


3) Hierarchical

Types of inheritance in java



4) Multiple



5) Hybrid

Types of inheritance in java

- Single Inheritance Example
- When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

Types of inheritance in java

- Single Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

```
barking...
eating...
```

Types of inheritance in java

- Multilevel Inheritance Example
- When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

Types of inheritance in java

- Multilevel Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Types of inheritance in java

- Hierarchical Inheritance Example
- When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Types of inheritance in java

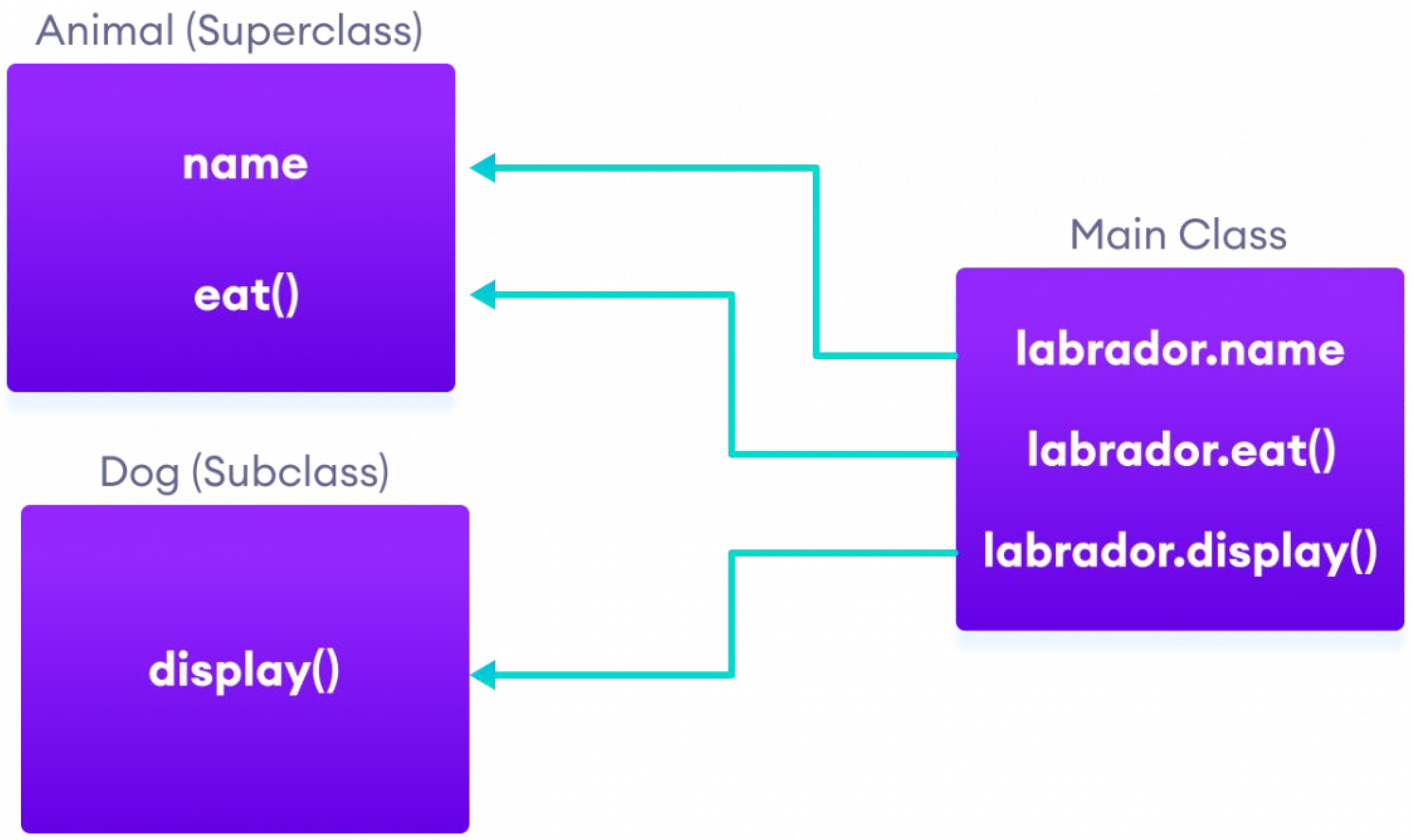
- Hierarchical Inheritance Example

```java
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Types of inheritance in java

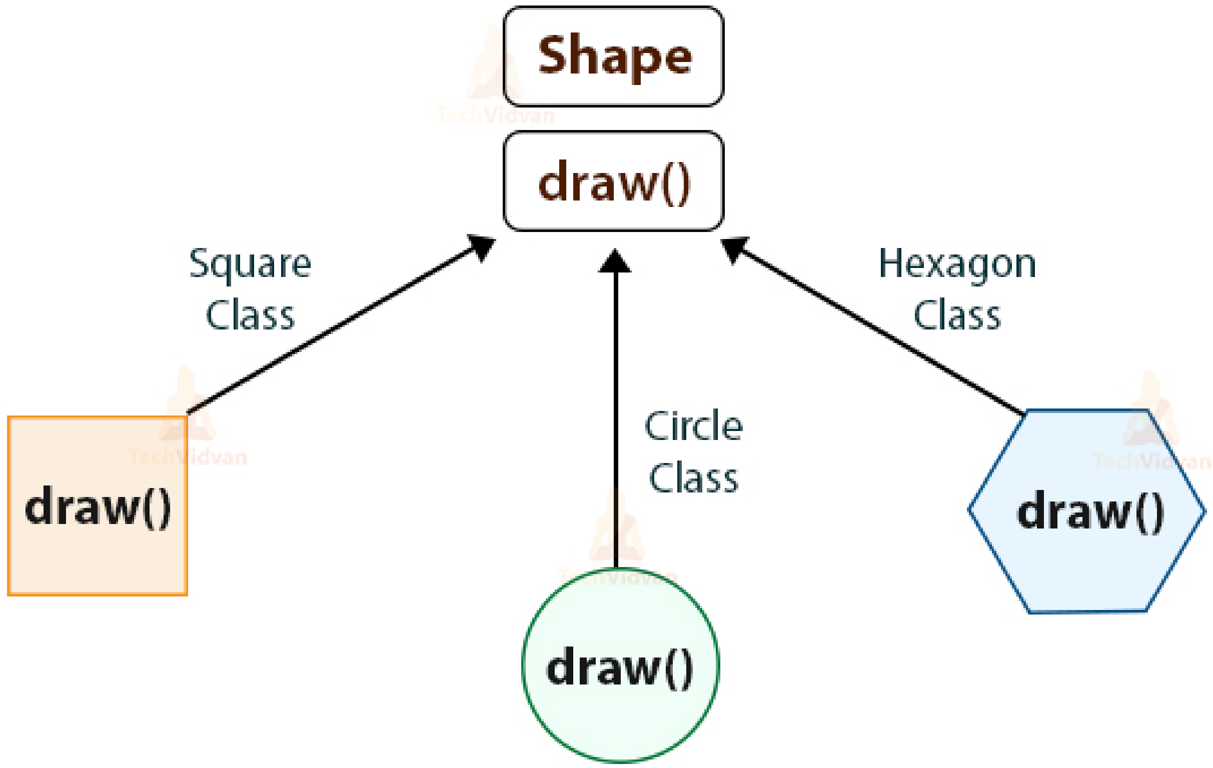


Java OOP Overriding

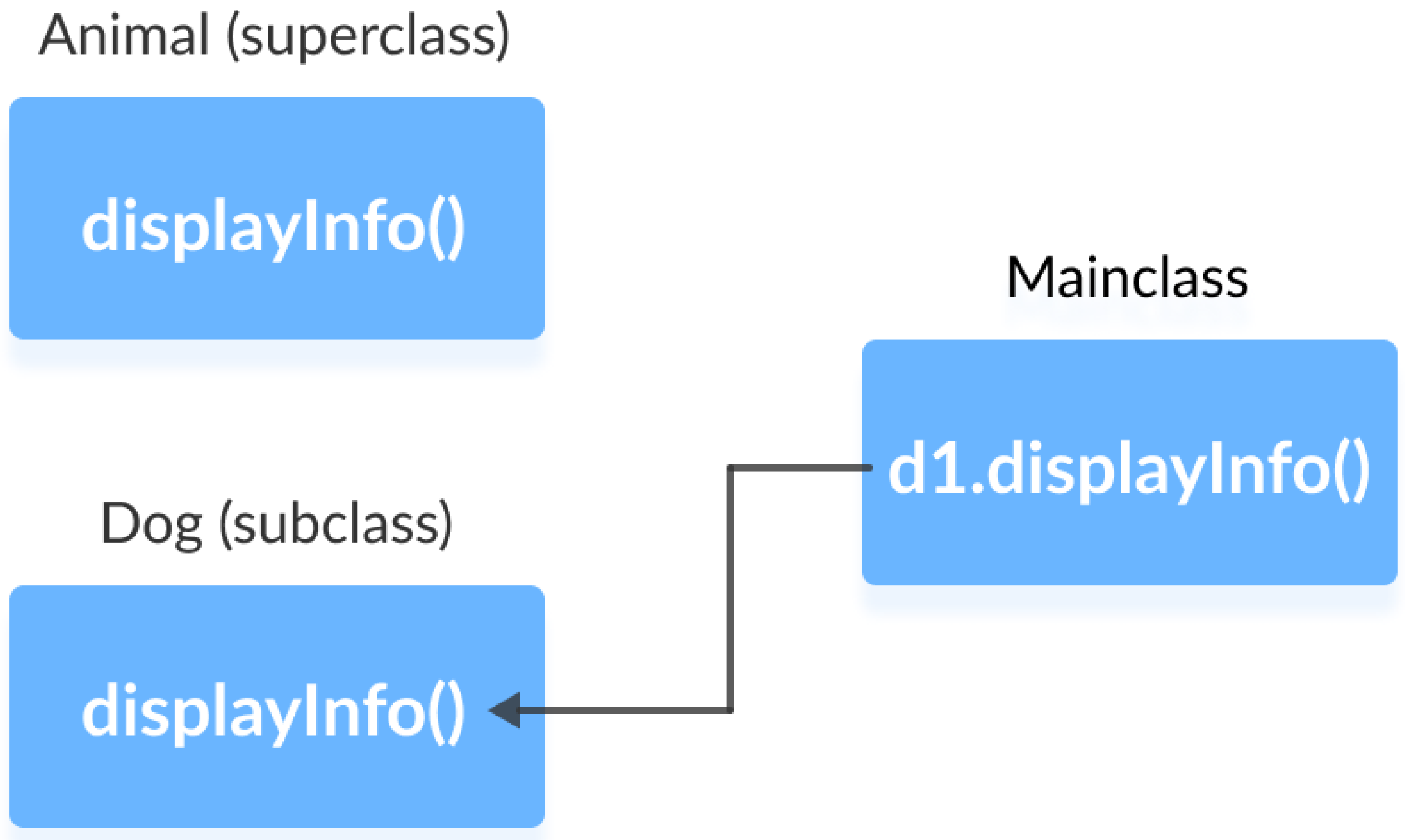
In Java, method overriding occurs when a subclass (child class) has the same method as the parent class. In other words, method overriding occurs when a subclass provides a particular implementation of a method declared by one of its parent classes.

Java OOP Overriding

Method Overriding in Java



Java OOP Overriding



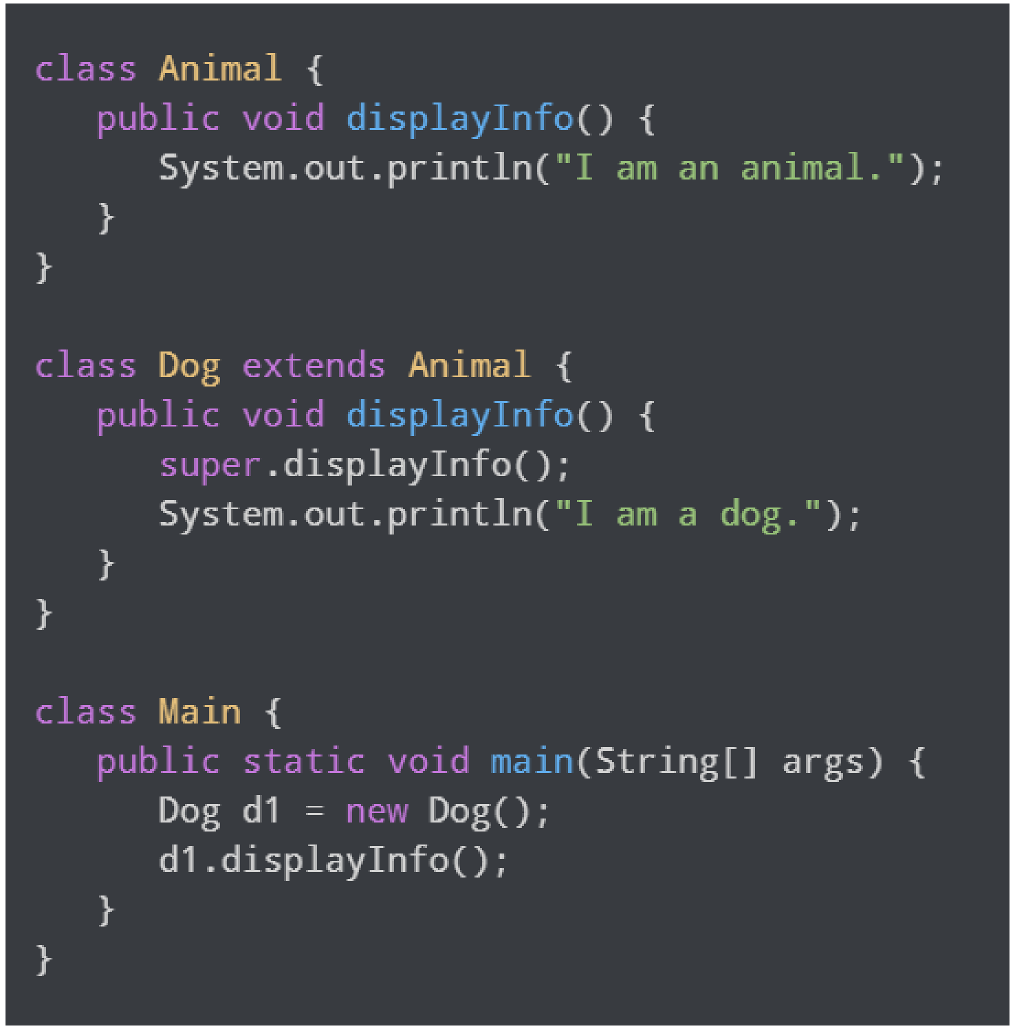
Java OOP Overriding

```java
class Animal {
   public void displayInfo() {
      System.out.println("I am an animal.");
   }
}

class Dog extends Animal {
   @Override
   public void displayInfo() {
      System.out.println("I am a dog.");
   }
}

class Main {
   public static void main(String[] args) {
      Dog d1 = new Dog();
      d1.displayInfo();
   }
}
```

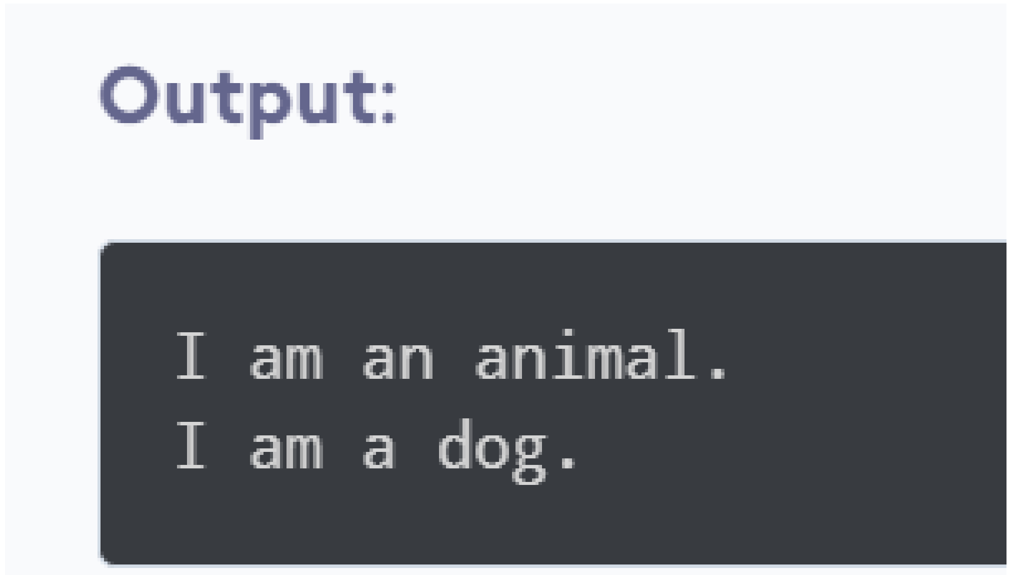
Output:

```
I am a dog.
```

Java OOP Overriding



```java
class Animal {
   public void displayInfo() {
      System.out.println("I am an animal.");
   }
}

class Dog extends Animal {
   public void displayInfo() {
      super.displayInfo();
      System.out.println("I am a dog.");
   }
}

class Main {
   public static void main(String[] args) {
      Dog d1 = new Dog();
      d1.displayInfo();
   }
}
```



Output:

```
I am an animal.
I am a dog.
```

Java OOP Overriding

```java
class Animal {
   protected void displayInfo() {
      System.out.println("I am an animal.");
   }
}

class Dog extends Animal {
   public void displayInfo() {
      System.out.println("I am a dog.");
   }
}

class Main {
   public static void main(String[] args) {
      Dog d1 = new Dog();
      d1.displayInfo();
   }
}
```

Output:

```
I am a dog.
```