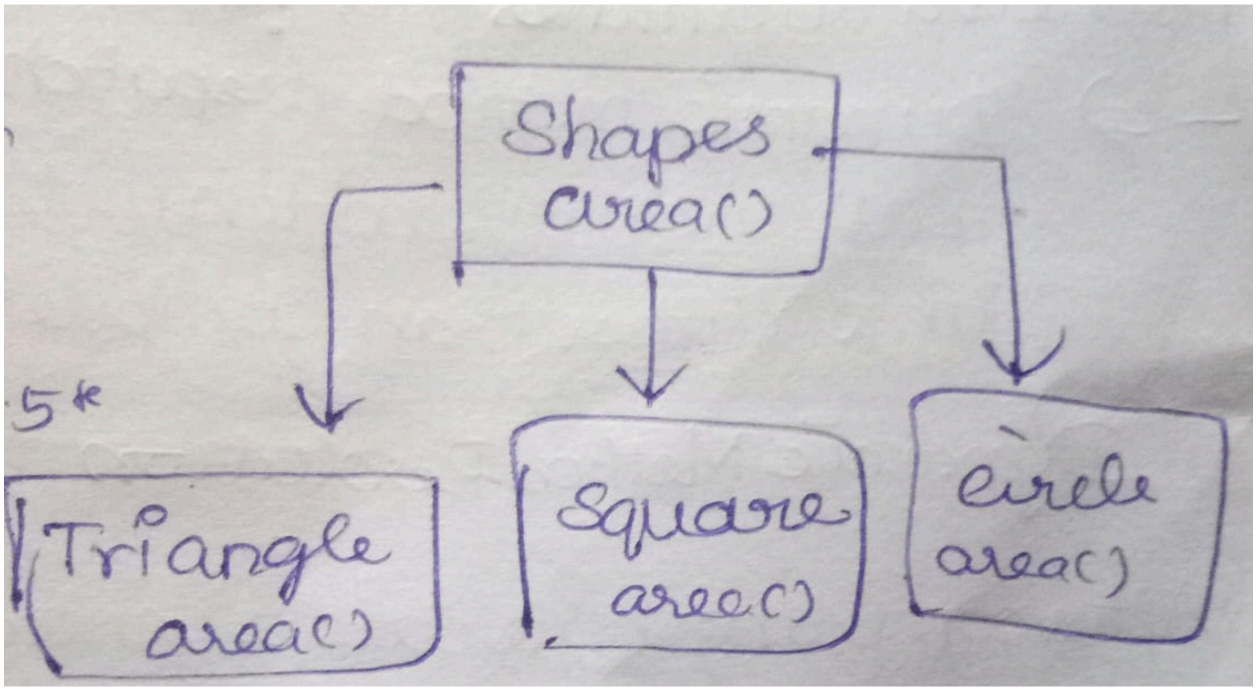
2️⃣

Polymorphism(Detailed notes)

2. POLYMORPHISM

Dynamic Method Dispatch is the process by which a call to an **overridden method** is resolved **at runtime** rather than compile time. It is a key feature of **runtime polymorphism** in Java.



Shapes.java(Parent Class)

```
package Polymorphism;
public class Shapes{
    void area(){
        System.out.println("I am in Shapes");
    }
}
```

Square.java(Overriding Class1)

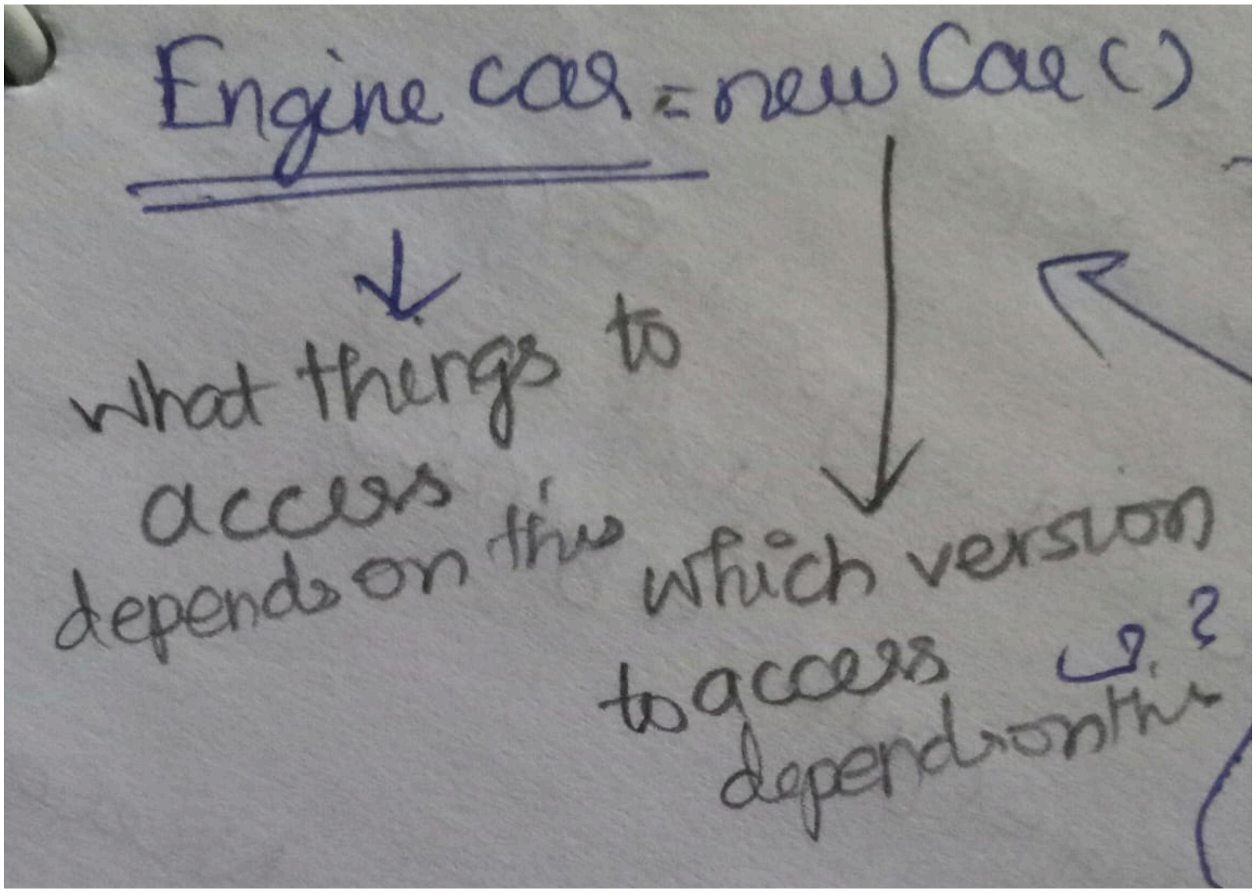
```
package Polymorphism;
public class Square extends Shapes{
    void area(){
        System.out.println("Area of square");
    }
}
```

Triangle.java(Overriding class2)

```
package Polymorphism;
public class Triangle extends Shapes{
    void area(){
      System.out.println("Area is 0.5*b*h");
    }
}
```

Circle.java(Overriding class2)

```
package Polymorphism;
public class Circle extends Shapes{
   @Override// Returns a error if not overriding
   void area(){
    System.out.println("Area of side : Pi*r*r ");
   }
}
```


Engine car = new Car()
what things to access depends on this
which version to access depends on this ?

TYPES OF POLYMORPHISM:

Main.java

```
package Polymorphism;
public class Main{
    public static void main(String[] args){
        Shapes shape=new Shapes();
        Circle circle=new Circle();
        Square square=new Square();

        // Usual call of appropriate method from Appropriate classes
        shape.area();
        circle.area();
        square.area();

        // Calling method of Squares from Shapes → Always access is determined
        //In runtime polymorphism, method call is determined at runtime, based on t
        Shapes square1=new Square();
        square1.area();

    }
}
```

Compile Time/Static Polymorphism

```
package Polymorphism;

// In this type..Java Determines from which class to access @compile time
public class CompileTimeExample{

    //Overloading based on Return type and type of parameters
    double sum(double a,int b){
        return a+b;
    }

    //Overloadig based on the datatype of the formal arguments
    double sum(int a,int b){
        return a+b;
    }

    // Overloading based on the number of parameters
    int sum(int a,int b,int c){
        return a+b+c;
    }

    public static void main(String[] args){
        CompileTimeExample obj=new CompileTimeExample();
        System.out.println("Sum using type of parameters: "+obj.sum(2,3));
        System.out.println("Sum using the number of parameters: "+obj.sum(4,5,6))
    }
}
```

RUNTIME POLYMORPHISM(Object Class)

```
package Polymorphism;
public class ObjectPrint{
    int num;
    public ObjectPrint(int num){
        this.num=num;
    }

    @Override
    public String toString(){
        return "ObjectPrint{"+num+"}";
    }

    public static void main(String[] args){
        ObjectPrint obj=new ObjectPrint(54);
        System.out.println(obj);
    }
}
```

Polymorphism using Static (Inheriting everything irrespective of the object called)

Box.java

```
package Polymorphism;
public class Box{
    public static void greeting(){
        System.out.println("Hey I am in Box");
    }
}
```

BoxWeight.java

```
package Polymorphism;
public class BoxWeight extends Box{
    public static void greeting(){
        System.out.println("Hey I am in BoxWeight!");
    }
}
```

Main.java

```
package Polymorphism;
public class Main1 {
    public static void main(String[] args) {
        Box.greeting();        // Calls Box's version
        BoxWeight.greeting();   // Calls BoxWeight's version

        //Returns only from Box and Not from the Box Weight as per the convention,
        // static method are already inherited ,irrespective of object is called or not
        Box box1 = new BoxWeight();
        box1.greeting();        // Still calls Box's version, not BoxWeight's!
    }
}
```

NOTE

DYNAMIC METHOD DISPATCH & DYNAMIC METHOD RESOLUTION

🔁 Dynamic Method Dispatch

Definition:

Dynamic Method Dispatch is the process in which a call to an overridden method is resolved at **runtime** rather than compile time. It is used to achieve **runtime polymorphism** in Java.

📝 **Key Point:**

The method that gets executed is determined by the **type of the object** (right-hand side) that the reference variable (left-hand side) is pointing to, **not** the reference type.

🧩 Dynamic Method Resolution

Definition:

Dynamic Method Resolution is the mechanism by which the **Java Virtual Machine (JVM)** determines which method to call at **runtime**, based on the actual object that is being referenced.

📝 **Key Point:**

This is the internal working concept of JVM behind **Dynamic Method Dispatch**.

💡 Summary

Concept	What it Does	When it Happens
Dynamic Method Dispatch	Calls overridden method based on object type	Runtime
Dynamic Method Resolution	JVM decides which method to link & invoke	Runtime