# INTRODUCTION TO JAVA

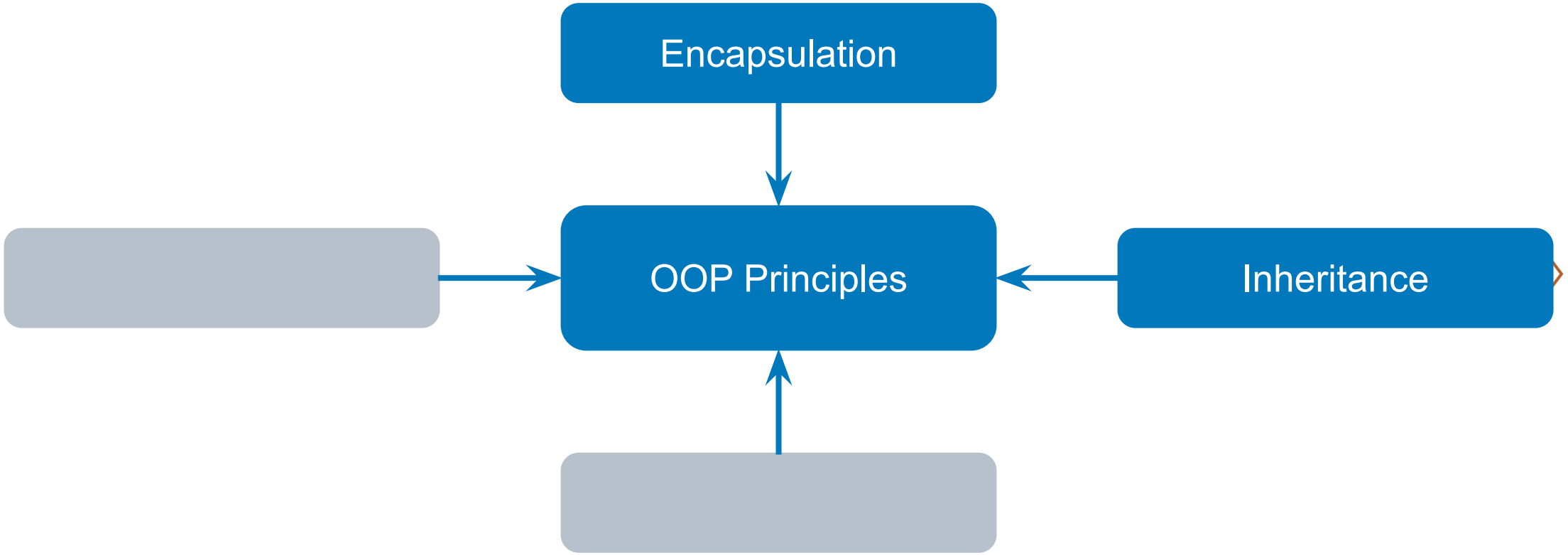
## Java 1.0

# INHERITANCE

## Lesson # 07

# OBJECT-ORIENTED PROGRAMMING PRINCIPLES

# PILLARS OF OBJECT-ORIENTED PROGRAMMING



# INHERITANCE

# INHERITANCE OVERVIEW

- The process by which one class acquires the properties (data members or fields) and behavior (methods) of another class is called inheritance
- The aim is to provide the reusability of code so that a class has to write only unique features

# INHERITANCE CONCEPTS

- Child class
  - The class that extends the features of another class is known as child class, subclass or derived class
- Parent class
  - The class whose properties and functionalities are inherited by another class is known as parent class, superclass or base class

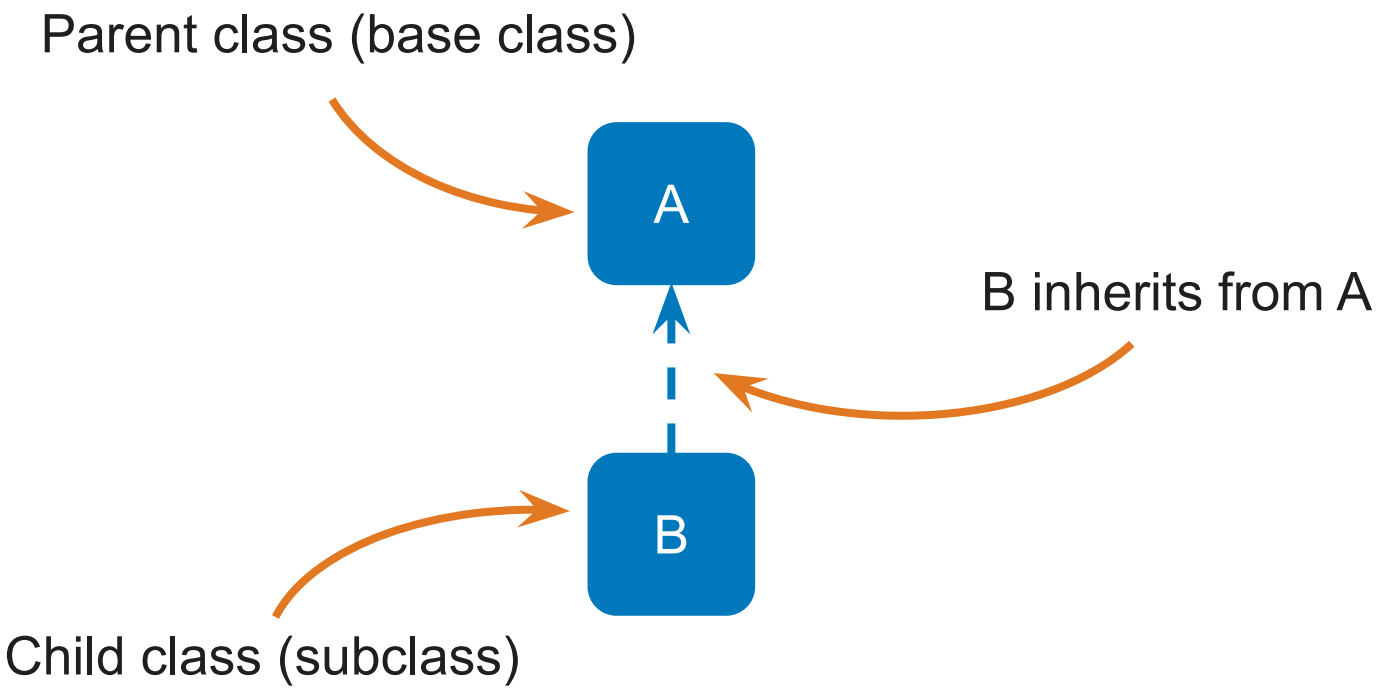
# JAVA TYPES OF INHERITANCE

- Single inheritance
  - Refers to a child and parent class relationship where a class extends the another class
- Multilevel inheritance
  - Refers to a child and parent class relationship where a class extends the child class

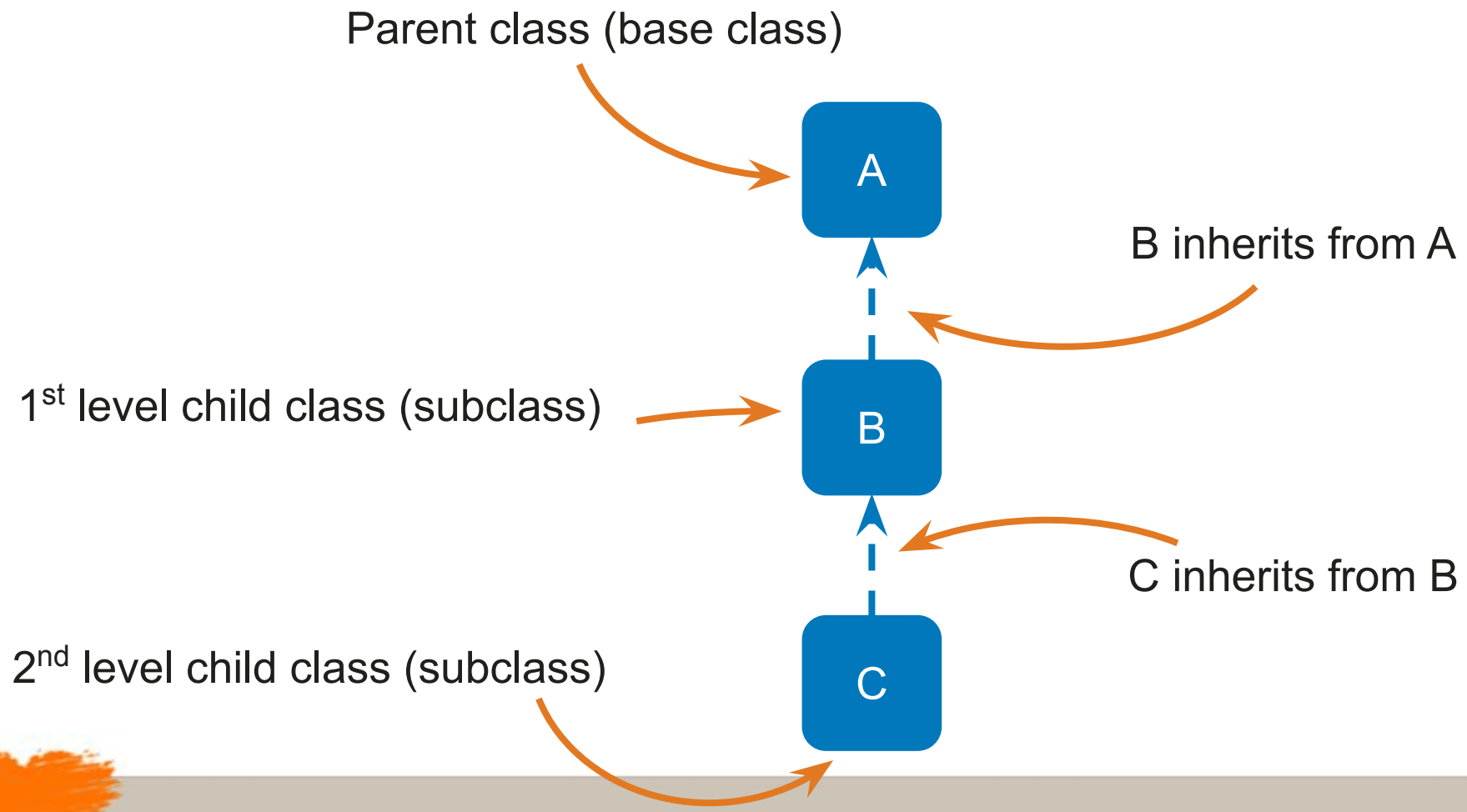
# JAVA TYPES OF INHERITANCE

- Hierarchical inheritance
  - Refers to a child and parent class relationship where more than one classes extends the same class
- Hybrid inheritance
  - Combination of more than one types of inheritance in a single program

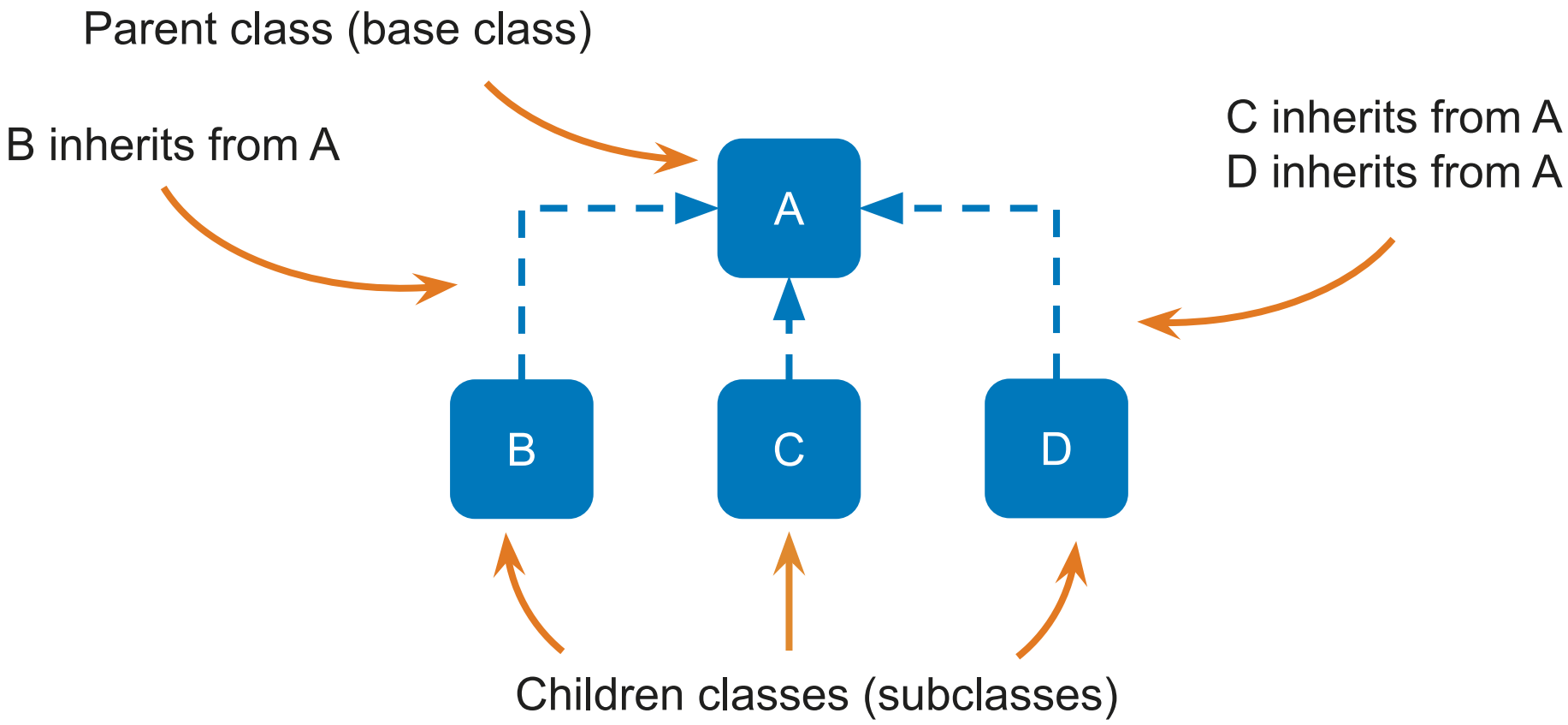
# SINGLE INHERITANCE



# MULTILEVEL INHERITANCE



# HIERARCHICAL INHERITANCE



# INHERITANCE EXAMPLE

Protected allows subclasses access fields or methods

```
public class Bicycle {

    protected String brand;
    protected int speed;

    public Bicycle(String brand, int speed) {
        this.brand = brand;
        this.speed = speed;
    }

    public void accelerate() {
        this.speed++;
    }

    public void decelerate() {
        this.speed--;
    }

    @Override
    public String toString() {
        return "Bicycle{" +
                "brand='" + brand + '\'' +
                ", speed=" + speed +
                '}';
    }
}
```

# INHERITANCE EXAMPLE

Subclass

Keyword stating inheritance process

Base class

Call parent's constructor

```
public class MountainBicycle extends Bicycle {

    protected int gear;

    public MountainBicycle(String brand, int speed, int gear) {
        super(brand, speed);
        this.gear = gear;
    }

    public void changeGear(int gear) {
        this.gear = gear;
    }

    @Override
    public String toString() {
        return "MountainBicycle{" +
                "gear=" + gear +
                ", brand='" + brand + '\'' +
                ", speed=" + speed +
                '}';
    }
}
```

# INHERITANCE EXAMPLE

Code

```
Bicycle bicycle = new Bicycle("Pinarello", 15);
MountainBicycle mountainBicycle = new MountainBicycle("BMC", 42, 2);

System.out.println(bicycle);
System.out.println(mountainBicycle);
```

Console output

```
Bicycle{brand='Pinarello', speed=15}
MountainBicycle{gear=2, brand='BMC', speed=42}
```

# INHERITANCE EXAMPLE

Code

```
System.out.println("Pedal to the metal!");
mountainBicycle.accelerate();

System.out.println(bicycle);
System.out.println(mountainBicycle);
```

Console output

```
Pedal to the metal!
 Bicycle{brand='Pinarello', speed=15}
 MountainBicycle{gear=2, brand='BMC', speed=43}
```

# JAVA INHERITANCE – RULES AND LIMITATIONS

- Every class has default implicit Object superclass
  - In the absence of any other explicit superclass, every class is implicitly a subclass of Object class
  - Object class has no superclass
- Single inheritance principle
  - A superclass can has any number of subclasses, but a subclass can have only one superclass
  - Multiple inheritance with interfaces is permitted, even though java does not support multiple inheritance with classes

# JAVA INHERITANCE – RULES AND LIMITATIONS

- Constructors are not inherited
- A subclass inherits all members (fields, methods, and nested classes) from its superclass
- Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass
- Private members inheritance
- A subclass does not inherit the private members of its parent class
- If superclass has public or protected methods (e.g. getters and setters) for accessing its private fields, these can also be used by subclass

# JAVA INHERITANCE – RECAP

- In subclasses we can inherit members as is, modify them, hide them, or supplement them with new members:
  - Use inherited fields directly, just like any other fields
  - Declare new fields in the subclass that are not in the superclass
  - Write a new method in the subclass that has the same signature as the one in the superclass, thus overriding it (e.g. equals(), toString())
  - Declare new methods in the subclass that are not in the superclass
  - Write a subclass constructor that invokes the superclass constructor, either implicitly or by using the keyword super

# REFERENCES

# REFERENCES


- https://stackify.com/oops-concepts-in-java/
- https://www.tutorialspoint.com/java/java_inheritance.htm
- https://beginnersbook.com/2013/03/oops-in-javaencapsulation-inheritance-polymorphism-abstraction/
- https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

# QUESTIONS?

# THANK YOU!