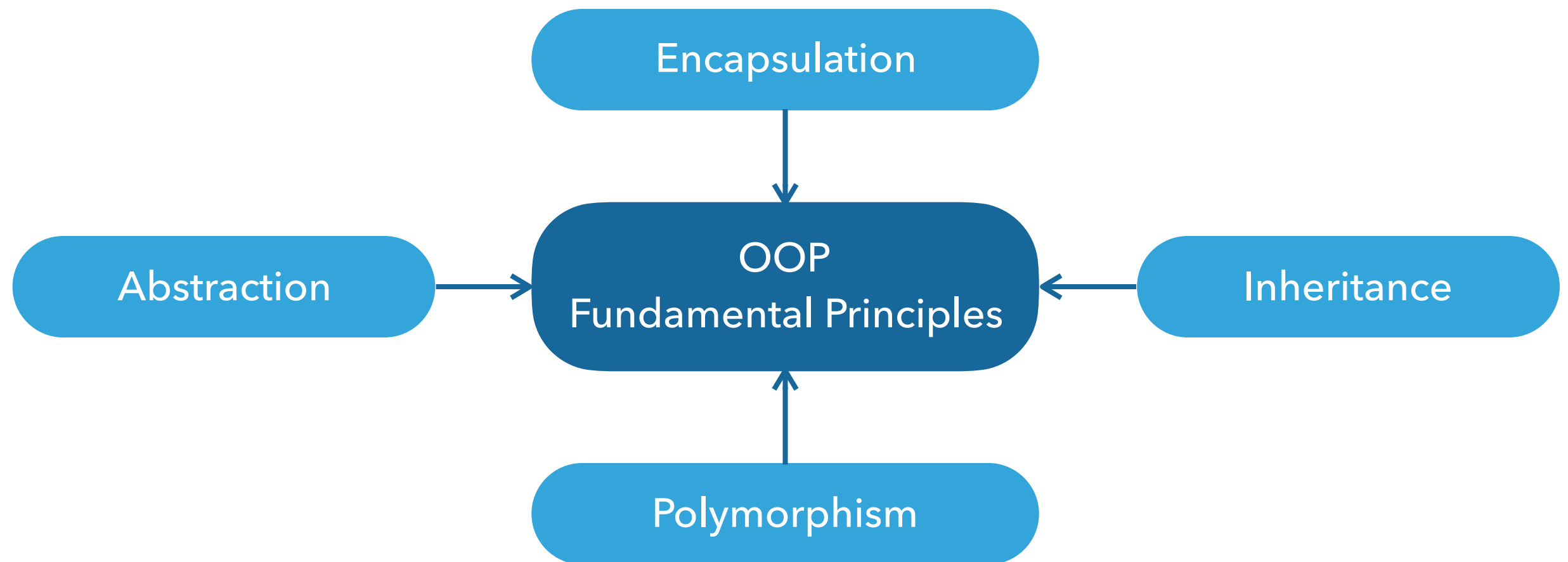JAVAGURU INTRODUCTION TO JAVA

LESSON 8

EXTENDED CONCEPTS OF
OBJECT ORIENTED PROGRAMMING

FOUR PILLARS OF OBJECT ORIENTED PROGRAMMING

INHERITANCE OVERVIEW

▸ The process by which one class acquires the properties (data members or fields) and behaviour (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
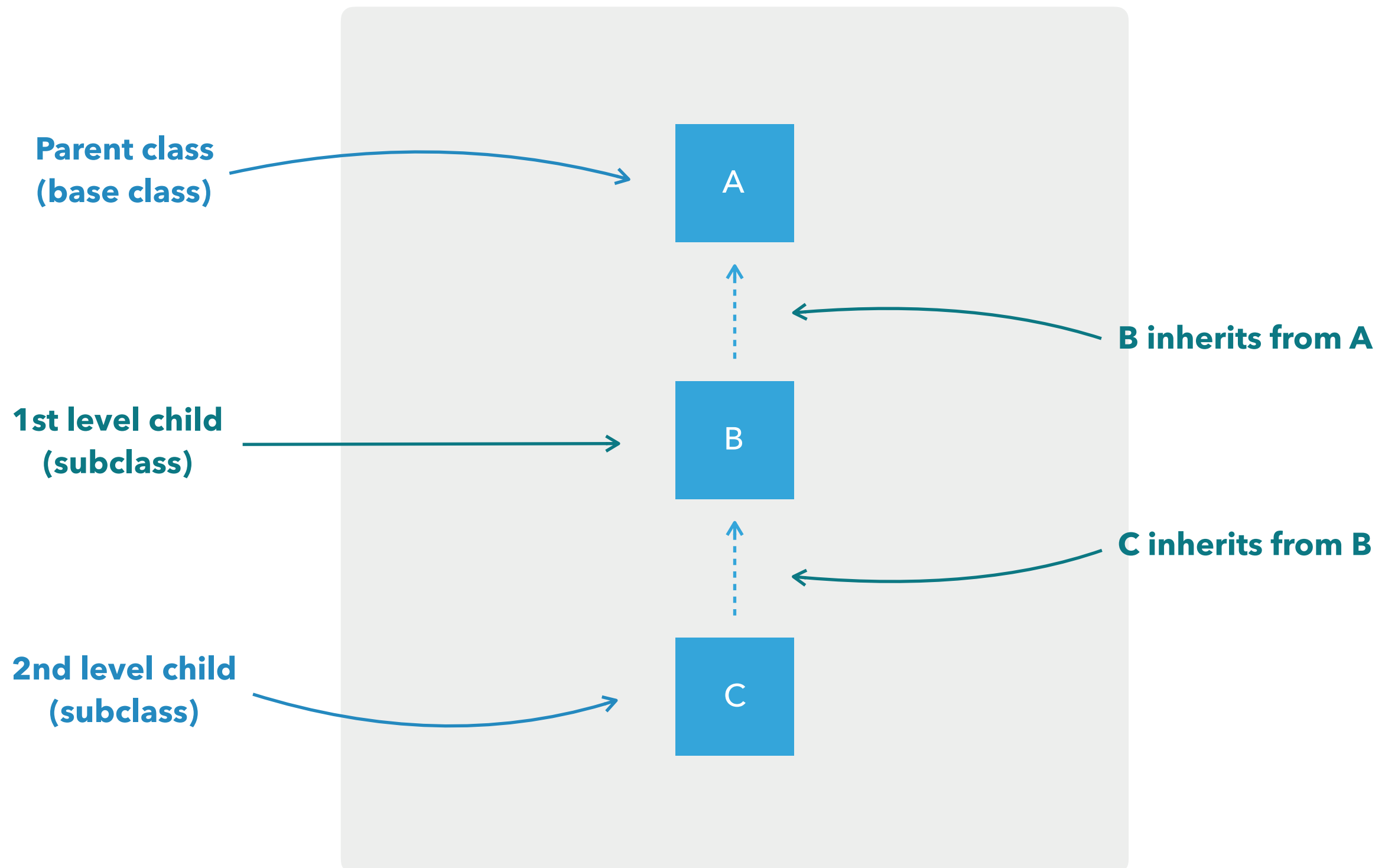
JAVA TYPES OF INHERITANCE: SUMMARY   IS-A

‣ 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

‣ 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

JAVA TYPES OF INHERITANCE: SINGLE INHERITANCE

Parent class (base class)

A

B inherits from A

B

Child class (subclass)
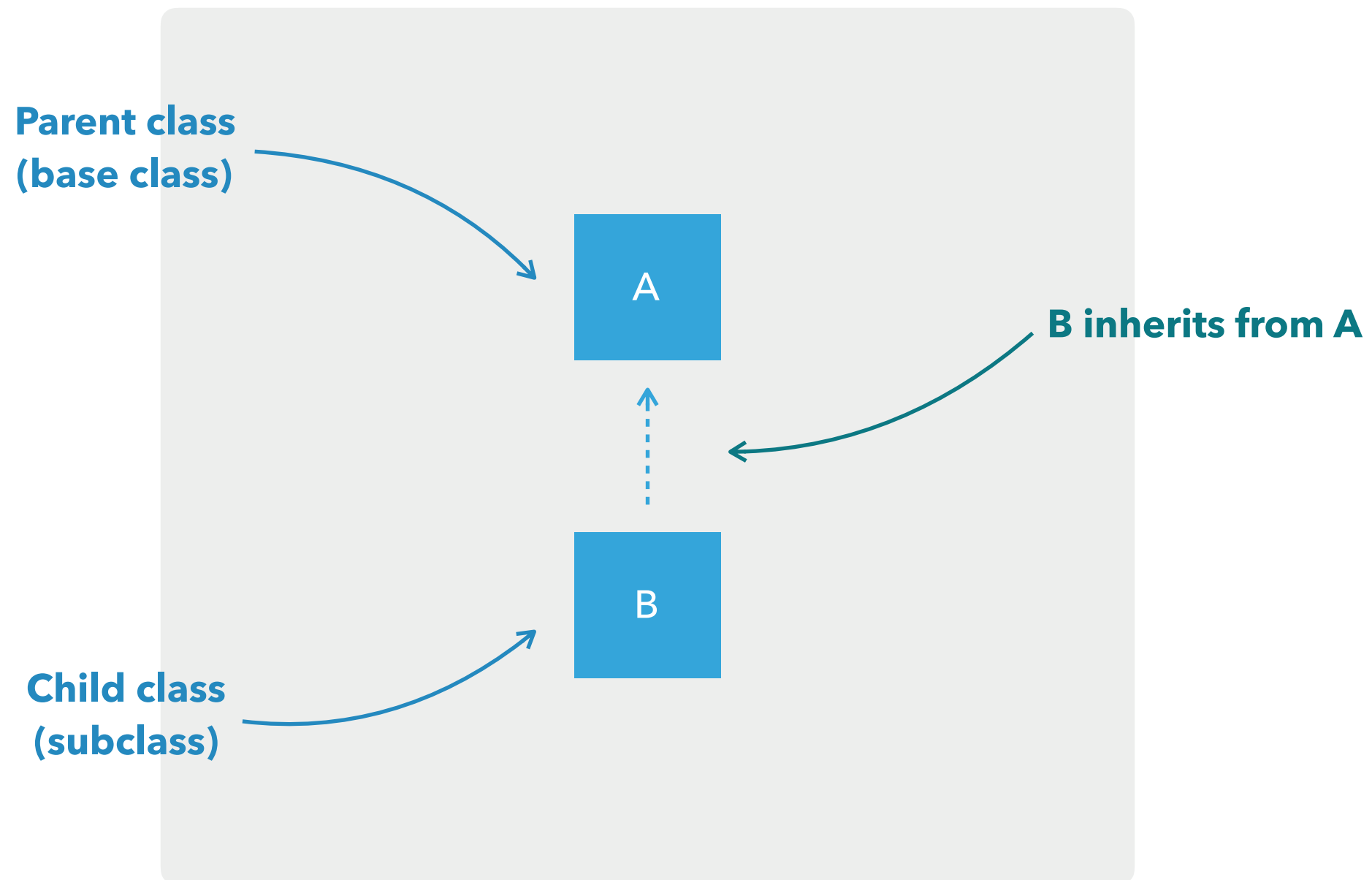
JAVA TYPES OF INHERITANCE: MULTILEVEL INHERITANCE
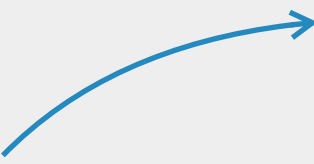
JAVA TYPES OF INHERITANCE: HIERARCHICAL INHERITANCE

Parent class (base class)

B inherits from A

A

D inherits from A, C inherits from A

B

C

D

Children classes (subclasses)
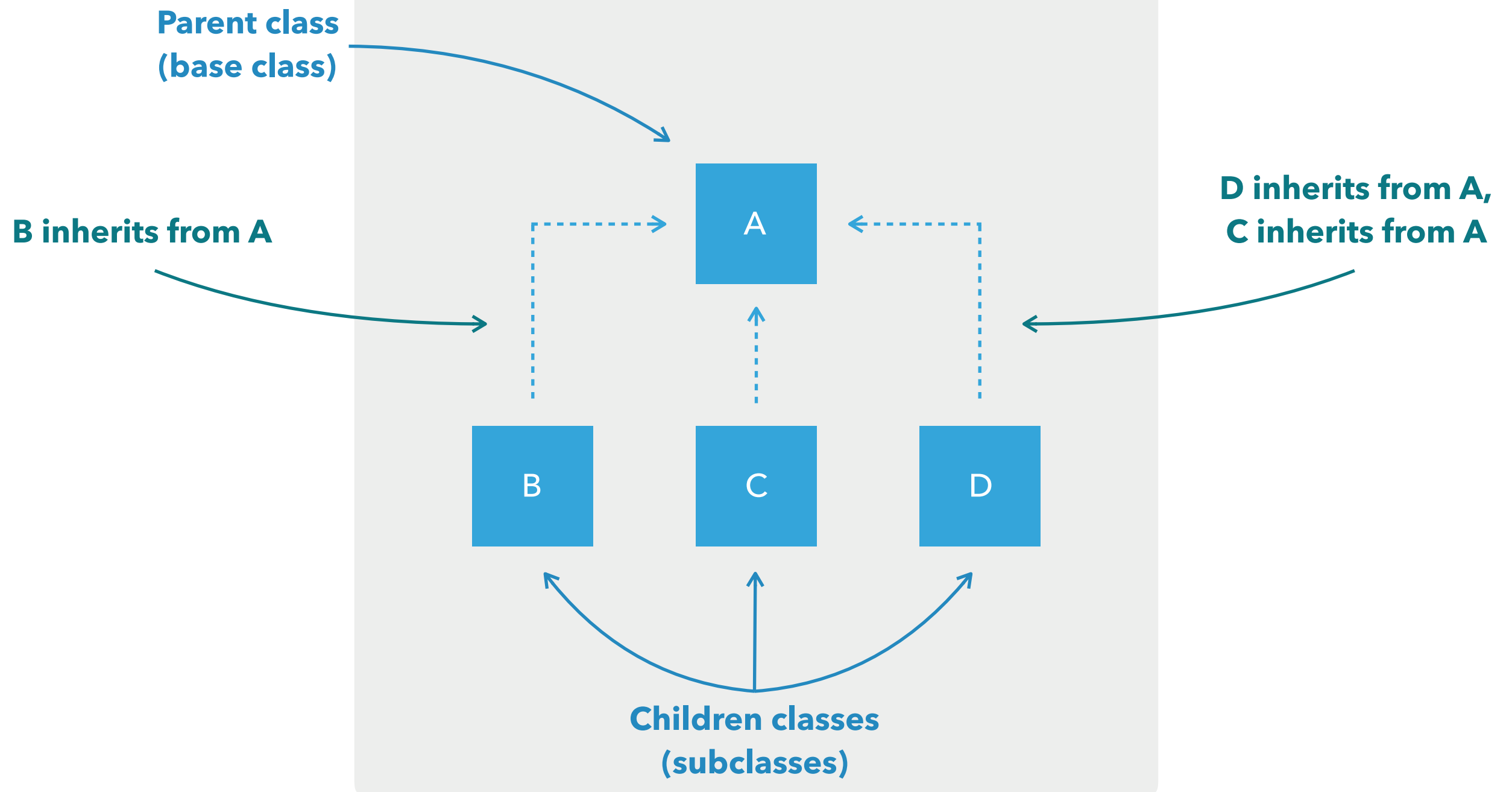
1. INHERITANCE: CODE EXAMPLE

Protected allows subclasses access fields or methods

```java
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 +
                '}';
    }

}
```

2. INHERITANCE: CODE EXAMPLE

Subclass

class ИмяПодкласса extends ИмяСуперкласса {}

Keyword stating inheritance process

Base class

Call parent's constructor

```java
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 +
                '}';
    }

}
```

3. INHERITANCE: CODE EXAMPLE

Code

```java
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}
```

4. INHERITANCE: CODE EXAMPLE

Code

```java
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}
```

1. 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

2. 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

ABSTRACTION OVERVIEW

▸ The process where you show only relevant data and hide unnecessary details of an object from user

▸ Allows you to abstract from usage and rather outline generic object functionality

▸ Defines what object does instead of how

JAVA ABSTRACTION: SUMMARY

▸ Abstraction is achieved by two mechanisms:

　▸ Interfaces

　　▸ Allows to achieve complete abstraction

　▸ Abstract classes

　　▸ Allows to achieve partial abstraction

JAVA ABSTRACTION: INTERFACES OVERVIEW

▸ A bit like class, except:

  ▸ Interface can only contain method signatures and fields

▸ Methods defined in interfaces cannot contain the implementation of method, only signature (return type, name, parameters, exceptions)

▸ Describes an object by actions it can perform

  ▸ Sometimes interface names end with '-able' postfix (e.g. comparable)

1. JAVA ABSTRACTION: INTERFACE CODE EXAMPLE

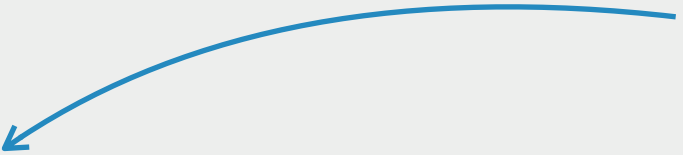**Interface keyword instead of class**

Interface name

```
public interface Singer {

    void sing();

}
```

Singers can sing, but we don't care how they do so

2. JAVA ABSTRACTION: INTERFACE CODE EXAMPLE

Special keyword to guarantee that we support interface specified behaviour

```java
public class ElvisPresley implements Singer {

    ...

    @Override
    public void sing() {
        System.out.println("Love me tender, baby..");
    }

    ...

}
```

Concrete implementation of singers behaviour

3. JAVA ABSTRACTION: INTERFACE CODE EXAMPLE

```java
public class MichaelJackson implements Singer {

    ...

    @Override
    public void sing() {
        System.out.println("Billie Jean is not my lover");
    }

    ...

}
```

4. JAVA ABSTRACTION: INTERFACE CODE EXAMPLE

```java
public class BritneySpears implements Singer {

    ...

    @Override
    public void sing() {
        System.out.println("Hit me baby one more time");
    }

    ...

}
```

JAVA ABSTRACTION: ABSTRACT CLASS OVERVIEW

▸ Mostly like a class, except:

  ▸ Can contain method signatures without implementation among other methods

  ▸ Cannot be instantiated

1. JAVA ABSTRACTION: ABSTRACT CLASS CODE EXAMPLE

Marking class as abstract

Subclasses must use constructor of parent class

Method signature that all children are forced to implement

```java
public abstract class Shape {

    private String color;

    public Shape(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    abstract double area();

}
```

2. JAVA ABSTRACTION: ABSTRACT CLASS CODE EXAMPLE

```java
public class Circle extends Shape {

    private int radius;

    public Circle(String color, int radius) {
        super(color);
        this.radius = radius;
    }

    @Override
    double area() {
        return 3.14 * radius * radius;
    }

}
```

Extending shape class with concrete details

Circle specific properties

Calling parent constructor with required params

Each concrete shape knows how to calculate its area
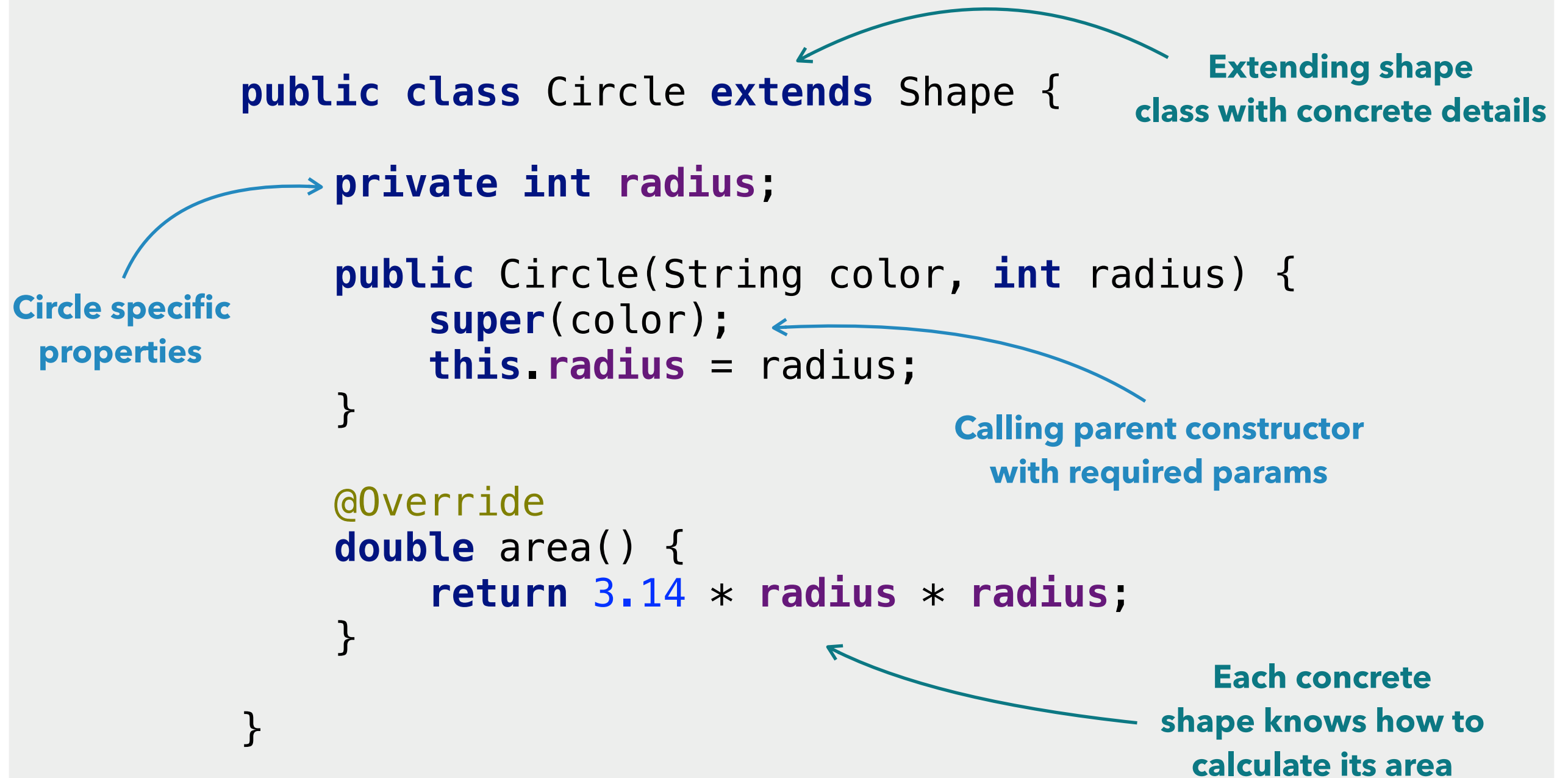
3. JAVA ABSTRACTION: ABSTRACT CLASS CODE EXAMPLE

```java
public class Rectangle extends Shape {

    private int width;
    private int height;

    public Rectangle(String color, int width, int height) {
        super(color);
        this.width = width;
        this.height = height;
    }


    @Override
    double area() {
        return width * height;
    }

}
```

Rectangle specific properties

1. JAVA ABSTRACTION: INTERFACE VS ABSTRACT CLASS

▸ Type of methods

  ▸ Interface can have only abstract methods (since Java 8 supports static and default methods as well)

  ▸ Abstract class can have abstract and non-abstract methods

▸ Final variables

  ▸ Variables declared in a Java interface are by default final

  ▸ Abstract class may contain non-final variables

2. JAVA ABSTRACTION: INTERFACE VS ABSTRACT CLASS

▸ Type of variables

  ▸ Interface has only static and final variables

  ▸ Abstract class can have final, non-final, static and non-static variables

▸ Implementation

  ▸ Interface can't provide the implementation of abstract class

  ▸ Abstract class can provide the implementation of interface

3. JAVA ABSTRACTION: INTERFACE VS ABSTRACT CLASS

▸ Inheritance vs Abstraction

  ▸ Interface can be implemented using keyword "implements"

  ▸ Abstract class can be extended using keyword "extends"

▸ Multiple Implementation

  ▸ Interface can extend another Java interface only

  ▸ Abstract class can extend another Java class and implement multiple Java interfaces

▸ Accessibility of data members

  ▸ Access modifiers of interface members are public by default and cannot be changed

  ▸ Access modifiers of abstract class members can have any access modifiers (except private abstract methods)

POLYMORPHISM OVERVIEW

▸ Polymorphism is the ability of an object to take on many forms

▸ Capability of a method to do different things based on the object that it is acting upon

▸ Which implementation to be used is decided at runtime depending upon the situation

1. POLYMORPHISM: CODE EXAMPLE

Code

```
Singer elvis = new ElvisPresley();
Singer jackson = new MichaelJackson();
Singer spears = new BritneySpears();

elvis.sing(); jackson.sing(); spears.sing();
```

Console output

Love me tender, baby..

Billie Jean is not my lover

Hit me baby one more time

2. POLYMORPHISM: CODE EXAMPLE

Code

```java
Singer[] singers = new Singer[2];
singers[0] = new ElvisPresley(); singers[1] = new BritneySpears();

for (Singer singer : singers) {
    singer.sing();
}
```

Console output

Love me tender, baby..

Hit me baby one more time

3. POLYMORPHISM: CODE EXAMPLE

Code

```java
Shape circle = new Circle("Red", 3);
Shape rectangle = new Rectangle("Blue", 2, 4);

System.out.println("Circle area = " + circle.area());
System.out.println("Rectangle area = " + rectangle.area());
```

Console output

```
Circle area = 28.259999999999998
Rectangle area = 8.0
```

REFERENCES

▸ https://stackify.com/oops-concepts-in-java/

▸ https://www.tutorialspoint.com/java/java_inheritance.htm

▸ https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/

▸ https://www.geeksforgeeks.org/abstraction-in-java-2/

▸ http://tutorials.jenkov.com/java/interfaces.html

▸ https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html