# Abstract Classes and Interfaces

## Abstract classes

- An abstract class is a generic class (or type) used define general characteristics of some related classes.
- An abstract class is always used as a superclass that cannot be instantiated (we cannot use it to create objects)
- An abstract class is declared using the keyword **abstract**.
- Subclasses that extend an abstract class will inherit all the abstract class's attributes and must implement any abstract method defined in the abstract class.

## Abstract Methods

- To declare an abstract method, you provide
  - the keyword **abstract**
  - the intended method type, name, and arguments
  - but you do not write the body of  the method
- Example:

  **abstract** int methodName(int x, int y);
- An abstract method must be overridden (implemented) in a subclass.

## Abstract Class Syntax

```
abstract class ClassName {
        …
        abstract Type Method1(args);
        …
        Type Method2()  {
            // method body
        }
}
```

1. Abstract method, no method body.

2. Must be implemented by subclasses

- When a class contains one or more abstract methods, it should be declared as abstract class.
- The abstract methods of an abstract class must be implemented in its subclasses.
- We cannot declare abstract constructors or abstract static methods.

# Abstract Class: Example

```
// Abstract class Person
public abstract class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public abstract void PrintInfo();
    public String getName() {
        return name; }
    public int getAge() {
        return age;    }
}// end Person
```

```
// Concrete class Teacher
public class Teacher extends Person
{
    private String subject;

 public Teacher(String name, int
age, String subject) {
        super(name, age);
        this.subject = subject;
    }

public void PrintInfo() {
System.out.println("I am a teacher.
My name is " + getName() + " and I
teach " + subject + ".");
    }
}
```

# Abstract Class: Example

```
// Concrete class Student

public class Student extends Person {

    private String college;

    public Student(String name, int age,
     String college) {

        super(name, age);

        this. college = college;

    }

    public void printInfo() {

 System.out.println("I am a student. My
     name is " + getName() + " and I am
     in grade " + college + ".");

    }

}
```

```
// Main class for testing
public class Main {
public static void main(String[] args)
{
Person teacher = new Teacher(“Ali",
35, “Java");

  teacher.printInfo();

 Person student = new
Student(“Alya", 18, "IT");
        student.printInfo();
    }
}
```

## Benefits of the abstract class

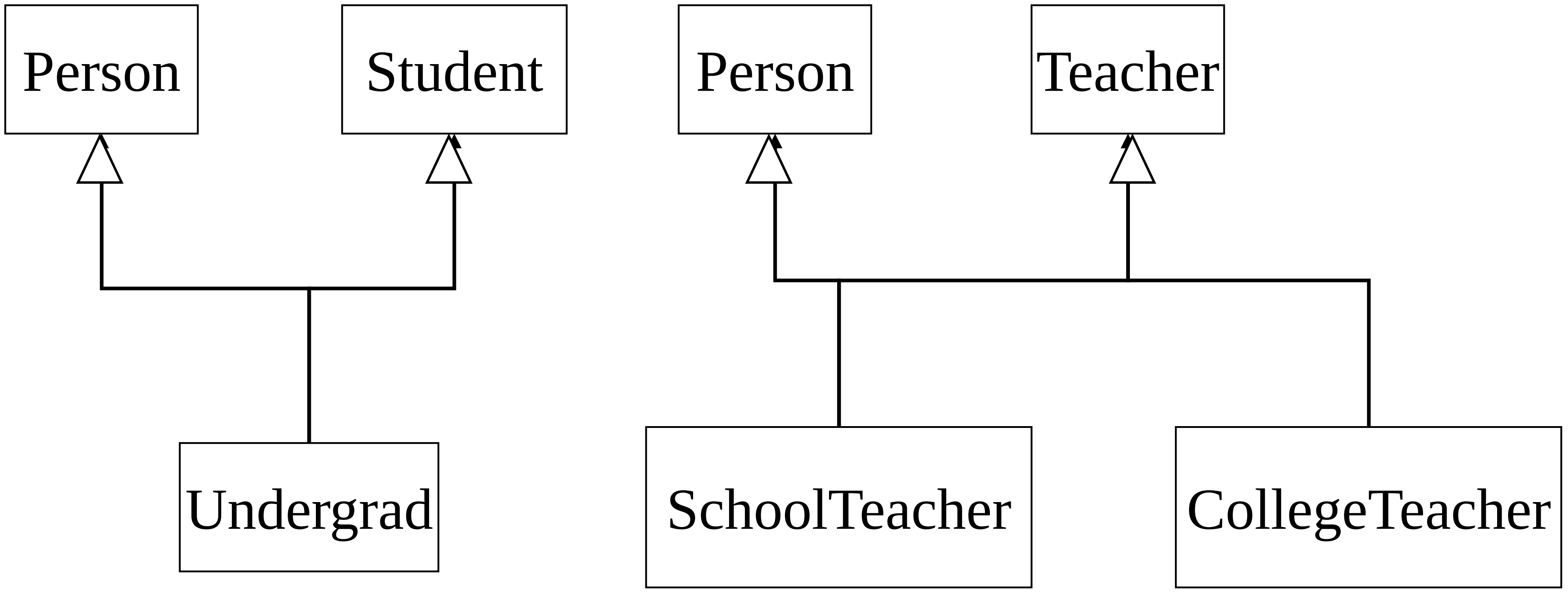
1. Code reusability: The abstract class Person provides common attributes and behaviors that are shared by both Teacher and Student. By defining these in the abstract class, we avoid duplicating code in the concrete classes.
2. Common interface: The abstract class defines the printInfo() method that is required to be implemented by the concrete subclasses. This enforces a contract, ensuring that both Teacher and Student have an printInfo() method.
3. Polymorphism: The abstract class Person allows objects of different concrete classes (Teacher and Student) to be treated uniformly through the common **Person** type. This promotes code flexibility and modularity, as methods that accept Person objects can work with any subclass of Person.
4. Future extensibility: By using an abstract class, you can easily add more concrete subclasses in the future, such as Administrator or Parent, without modifying existing code that works with the Person type.

## Interfaces in Java

- An interface is a class-like construct that contains only constants and abstract methods.
- In many ways, an interface is similar to an abstract class.
- Interfaces are used to solve multiple inheritance problems.

# Java does not allow Multiple inheritance

- Multiple inheritance means that a class inherits method from more than one parent class.



# Java Interface

To distinguish an interface from a class, Java uses the following syntax to declare an interface:

```
public interface InterfaceName {
    constant declarations;
    method signatures;
}
```

Example:

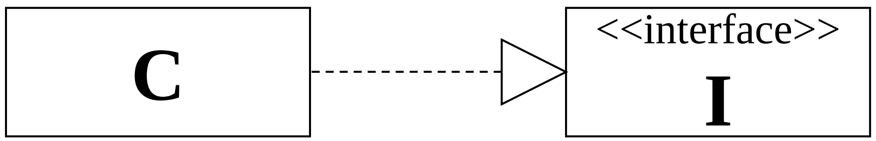
```
public interface Person {
    public abstract void printInfo();
    public abstract void getName();
    public abstract void getAge();    }
```

## Interface is a special class

- An interface is treated like a special class in Java.
- Each interface is compiled into a separate bytecode file, just like a regular class.
- Like an abstract class, you cannot create an instance from an interface using the <u>new</u> operator.
- But in most cases you can use an interface more or less the same way you use an abstract class. For example, you can use an interface as a data type for a variable.
- A variable of a Java interface type may contain objects of any class that implements the interface. (polymorphic assignment)

## To implement an interface

- ❑A class formally implements an interface by
  1. stating so in the class header.
  2. providing implementations for each abstract method in the interface.
- ❑If a class asserts that it implements an interface, it must define all methods in the interface or the compiler will produce errors.
- ❑If a class implements an interface, it establishes a realization relationship (one kind of inheritance relationship) between the class and the interface.



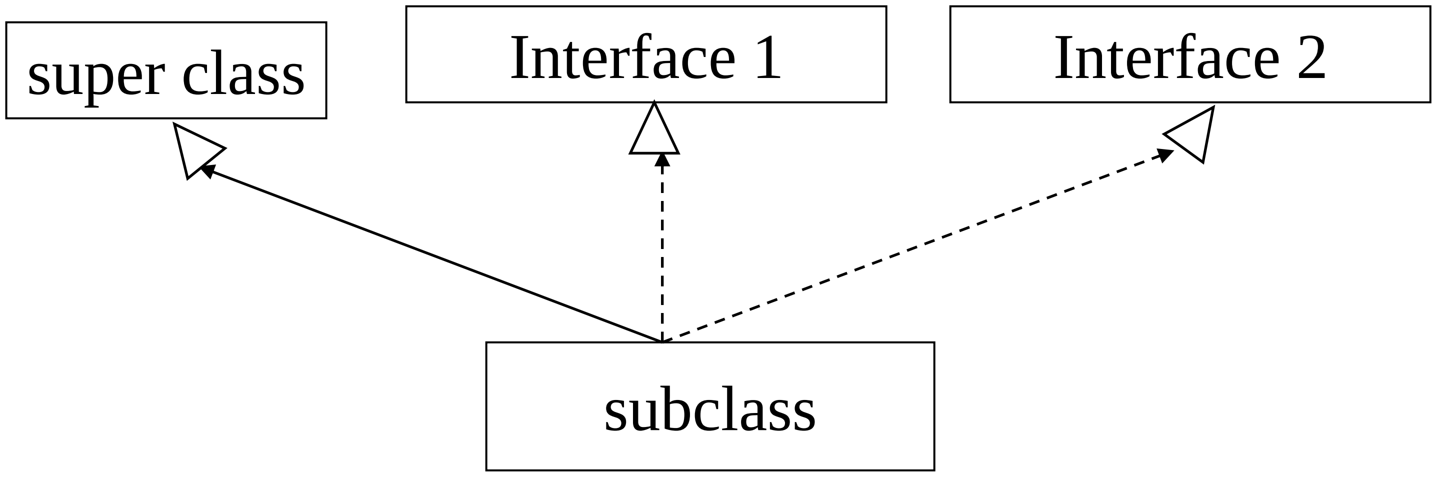
## Example 1

```
public class C implements I {
   public void method1 () {
      // some code.
   }

   public void method2 () {
      // some code
   }

   …etc.
}
```

Each method listed in interface I is given a definition

# Implement multiple interfaces

- A class can extend only **one parent** class.
- A class can implement **multiple interfaces**.
- The interfaces are listed in the implements clause, separated by commas.
- The class must implement all methods in all interfaces listed in the header.



# Example

```
public class C implements I1, I2 {
    public void method1 () {
        // some code.
    }

    public void method2 () {
        // some code
    }

    …etc.
}
```

Each method listed
in interface I1 and I2 is
given a definition

## Interface Example

```
// Interface for a person
public interface Person {
    String getName();
    int getAge();
    void printInfo();
}
```

```
// Teacher class implementing the Person interface
public class Teacher implements Person {
    private String name;
    private int age;
    private String subject;
    public Teacher(String name, int age, String subject) {
        this.name = name;
        this.age = age;
        this.subject = subject;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void introduce() {
        System.out.println("I am a teacher. My name is " + 
getName() + " and I teach " + subject + ".");
    }
}
```

# Interface Example

```
// Student class implementing the Person interface

public class Student implements Person {

    private String name;

    private int age;

    private String college;

    public Student(String name, int age, String
      college) {

        this.name = name;

        this.age = age;

        this.college = college;   }

    public String getName() {return name; }

    public int getAge() { return age;    }

    public void printInfo() {

        System.out.println("I am a student. My name is
      " + getName() + " and I study at " + college +
      ".");

    }}
```

```
// Main class for testing
public class Main {
public static void main(String[] args)
{
Person teacher = new Teacher(“Ali",
35, “Java");


  teacher.printInfo();


 Person student = new
Student(“Alya", 18, "IT");
        student.printInfo();
    }
}
```