# CS202: IT Workshop
# Java

# Abstract class, Interface

**Ref**:

1. Harvey Deitel, Paul Deitel, **Java: How to Program**, 9/e, Prentice Hall India.
2. Herb Schildt, **Java: The Complete Reference**, 8/e Tata Mcgraw Hill Education.

# What have we seen in last lecture?

- Inheritance: Code reusability

```
class Student extends Person {
. . . . .
}
```

```
Person p = new Person();
p.getDetails();

Student s = new Student();
s.getDetails();

Faculty f = new Faculty();
f.getDetails();
```

- **Polymorphism**: Method overriding, overloading
- Constructors in super and sub class

```
class Student extends Person {
   . . .
      Student (String name, int age, int roll) {
            super( name, age );
            . . .
            }
    . . .
```

```
double distance ( int x1, int y1) {
            . . .
   }
double distance (Point p) {
            . . .
   }
```

```
p.distance(2,3); p.distance (q);
```

# Polymorphism in Java: Abstract method

- When the definition / implementation of a method is **fully subclass specific**,
  - Declare the method in the superclass
  - Define / implement in the respective subclasses

- Method in the superclass is called **abstract method**

- But why to **declare** in super class?

```
public abstract void findExceptional();
```

***findExceptional*() has no work to do in class Person; BUT it has significance!**

**For student, cpi > 9.5**
**For faculty, noOfPub > 1000**

# Abstract class

- Abstract method is to be declared inside an **abstract class**
- An abstract class is one which **cannot** have any object
- Abstract class may contain other methods (non-abstract / concrete) too
- We should put all the **common-essential features** (application specific) in an abstract class

```
abstract class Person {
        . . .
        void printDetails() {
                System.out.println ("name: " +name+ "age: "+age);
        }
        public abstract void findExceptional();
}
```



Code (AbstractDemo.java): Screen share

# Abstract class

- When we extend an abstract class, we must implement **all** the abstract methods of the class

```
abstract class A {
. . .
abstract void f1();
abstract void f2();
void f3() { . . . }
}
```

```
class B extends A {
. . .
void f1() { . . . }
void f2() { . . . }

}
```

- Or, we need to declare the subclass as **abstract** too

```
abstract class A {
. . .
abstract void f1();
abstract void f2();
void f3() { . . . }
}
```

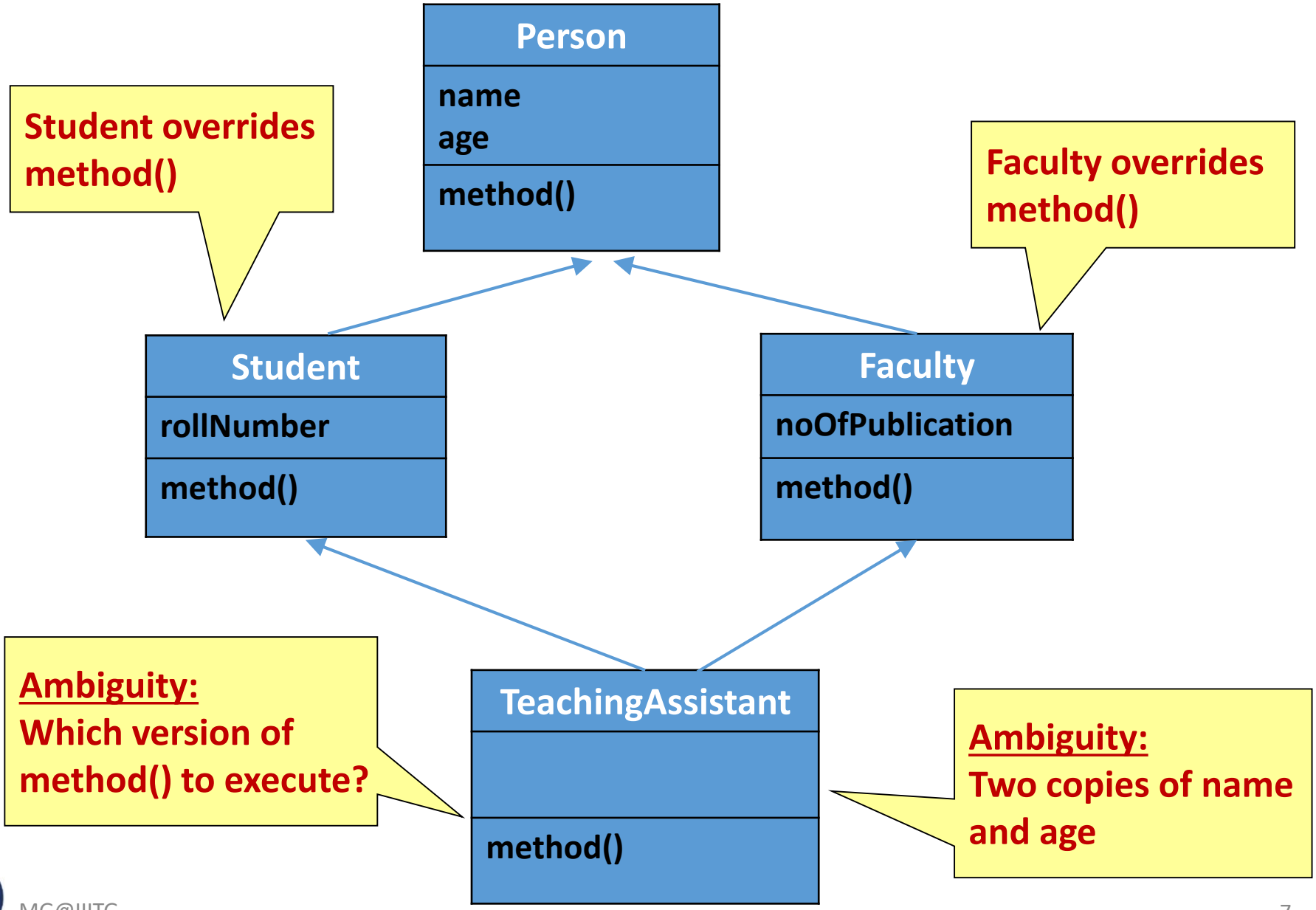
```
abstract class B extends A {
. . .
abstract void f1();
void f2() { . . . }

}
```



Code (AbstractDemo.java): Screen share

# Questions?

# Inheritance: Diamond Problem



# Interface: A special support from Java

- **No multiple inheritance** (more than one super class) for classes
- No instance variables; only Constants are declared as data members (**Interface**)
- No method body is present (**Interface**)

```
interface DisplayObject {
        int constant=5;
        public void display(String str1, String str2);
}
```

Only constants and to be initialized

```
class DisplayLine implements DisplayObject {
        . . .
        public void display(String str1, String str2) {
                . . .
        }
}
```

We can have additional fields in the class

We must define the method

# Interface in Java

- When we completely make a class abstract, we call it an **Interface**
- A class needs to **implement** an interface
  - Writes body of all the methods
  - We can create objects of the class
- A class may implement multiple interfaces
- An interface can **extend** interface(s)

- **Hide** all the details and just provide some **interfaces** to the outside world (a framework/blueprint)
- Whoever wants to use the features may **implement** the interface

**Code (**DisplayDemo.java, InterfaceTest.java**): Screen share**

# Questions?

# Final keyword in Java

- **Method overriding** allows to modify the definition of a method (a good feature of OOP)
- When we **do not** want any subclass to modify any method,
  - **Declare the method in the superclass as final.**
  - What about private methods?

  (Private methods cannot be overridden; thus **implicitly final**)

- When we **do not** want a class to be inherited, we can make it **final**.

- All **methods** in a final class are implicitly final.

# Final keyword in Java

- Class, method or variable can be declared final (**depending on its usage**).

| Item | Effect |
|---|---|
| **final** Class | Cannot be extended |
| **final** Method | Cannot be overridden |
| **final** Variable | Values cannot be changed |

```
final class Alumni {
        . . .
}
```

```
final void isAdult() {
        . . .
}
```

```
final double rateOfInterest = 3.5;
```

- **final variables need to be initialized in declaration or using constructor**

# Questions?