CPE207 Object Oriented Programming

Week 11

Interfaces:

Abstraction, Multiple inheritance

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

These Slides mainly adopted from Assist. Prof. Dr. Ozacar Kasim lecture notes

Typecasting

- **Reference type casting**
  - Casting from a subclass to a superclass is called **upcasting**
    - `Cat cat = new Cat();  // class Cat extends Animal`
    - `Animal animal = (Animal) cat;`
  - Casting from a superclass to a subclass is called **downcasting**
    - `Animal animal = new Cat()`
    - `((Cat) animal).meow(); //` extends the method available to Cat object
- **Upcasting** narrows methods and attributes available to this object,
- **Downcasting** extends methods and attributes available to this object

Exercises: Followings will compile? Run?

Week 11

```
Cat cat = new Animal();      //??

- Animal a = new Cat();
  - a.meow();                 //??
  - ((Cat)a).meow();          //??
- Object o = new Animal();
  - o.eat();                  //??

  - o.meow();                 //??
  - ((Animal)o).eat();        //??
  - ((Animal)o).meow();       //??
  - ((Cat)o).eat();           //??   (*)
  - ((Cat)o).meuw();          //??
```

```
public class Animal {

    protected String color;

    void eat(){
        System.out.println("animal eating...");
    }
}
```

```
public class Cat extends Animal {
    int age;

    @Override
    void eat() {
        System.out.println("cat is eating");
    }
    void meow(){}
}
```

* It is legal to cast a reference to the wrong subtype;
However, this will compile but crash when the program runs

Interfaces: Definition

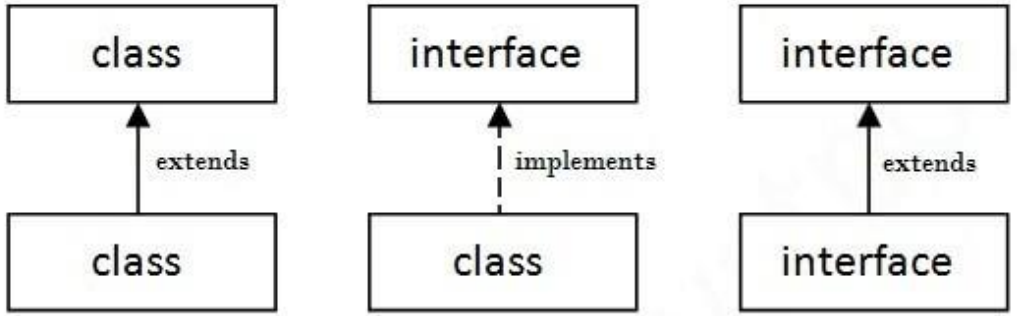
- An interface is like an abstract class that is intended to be used as a common base to access a number of similarly-behaving objects.

- An interface is a **contract** an implementing class needs to respect
  - interface contains behaviors that a class implements. (Contract)
  - Methods in an interface are by-default public. (Contract)

Interfaces : Introduction

- There are mainly three reasons to use interface. Using interface we can:
  1. achieve abstraction.
  2. support the functionality of multiple inheritance.
  3. achieve loose coupling.

Relationship between classes and interfaces



We already know this!

Interfaces: Syntax and Structure

- An interface is created with the following syntax:

```
modifier interface InterfaceID {
  //constants
  //abstract method
}
```

- An interface can extend other interfaces with the following syntax:

```
modifier interface interfaceID extends interface1, .., interfaceN {
   //constants
  //abstract method, (After Java 8.0 static and default methods)
}
```

- It provides total abstraction which means
  - all the methods in an interface are declared with the empty body,
  - and all the attributes are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

Interfaces: More …

1. Interfaces are like abstract classes.
2. An interface is not extended by a class; **it is implemented by a class.** But, one interface can extend another one.
3. An interface does not contain any constructors.
4. A class can implement any number of interfaces. **(multiple inheritance)**
5. Interfaces cannot be instantiated. (like abstract classes)
6. You can use interface as a data type for variables. (Like classes)
7. Interfaces can contain only abstract methods and constants.

Interface vs. abstract class

	Interface	Abstract class
Variable (attribute)	Only constants	Constants and variable data
Methods (behaviour)	No implementation allowed (no abstract modifier necessary)	Abstract or concrete

Interface vs. abstract class (cont)

Interface	Abstract class
A subclass can **implement** many interfaces	A subclass can inherit only one class
Can extend numerous interfaces	Can implement numerous interfaces
Cannot extend a class	Extends one class

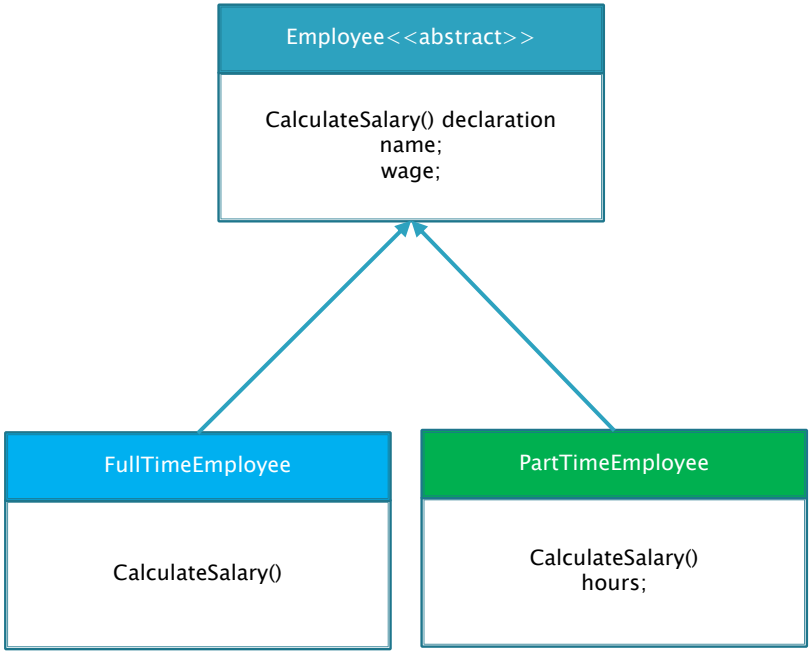
Interface usages

Abstraction

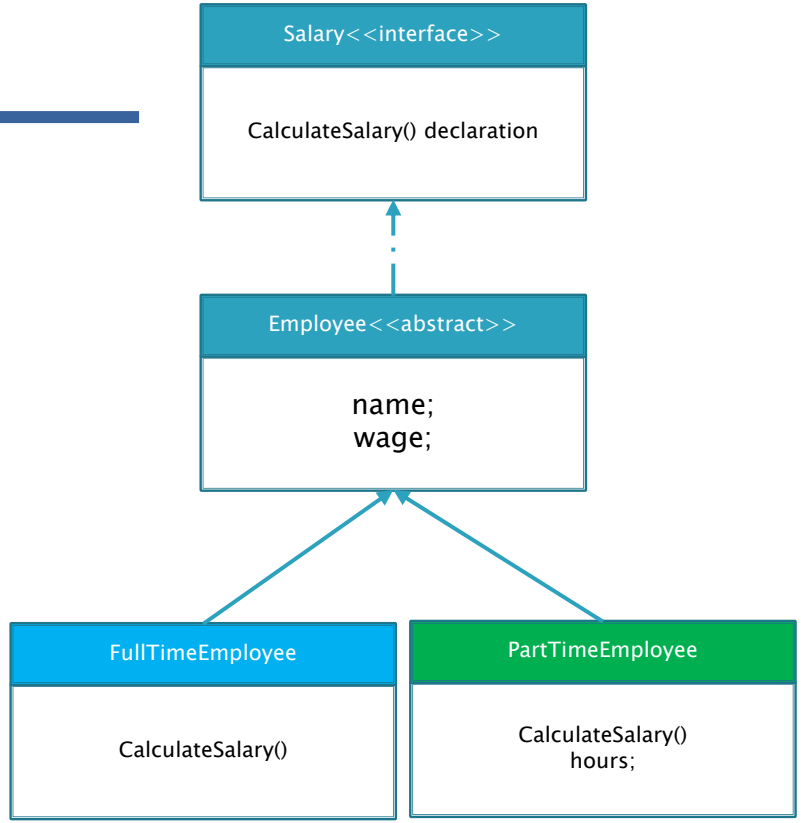
- Abstract classes vs. Abstraction
  - ***Abstract classes*** are used for inheritance, which is more heavily used to achieve code reuse
  - ***Abstraction*** enables large systems to be built without increasing the complexity of code and understanding.

  - In other words, abstraction is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user. We do abstraction when deciding what to implement.
  - We achieved abstraction using **abstract classes** and **interfaces**.

Example



Abstraction using Abstract classes

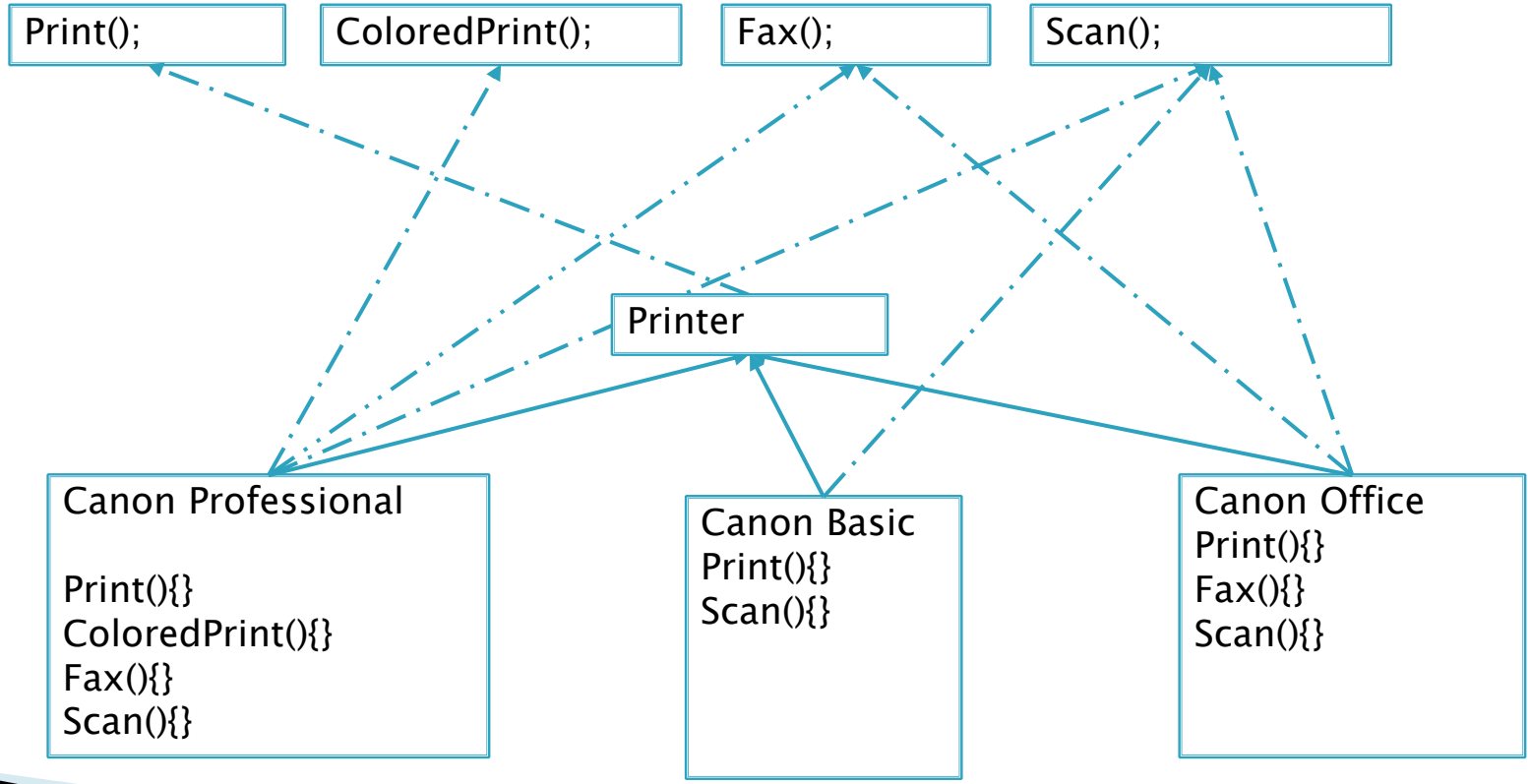


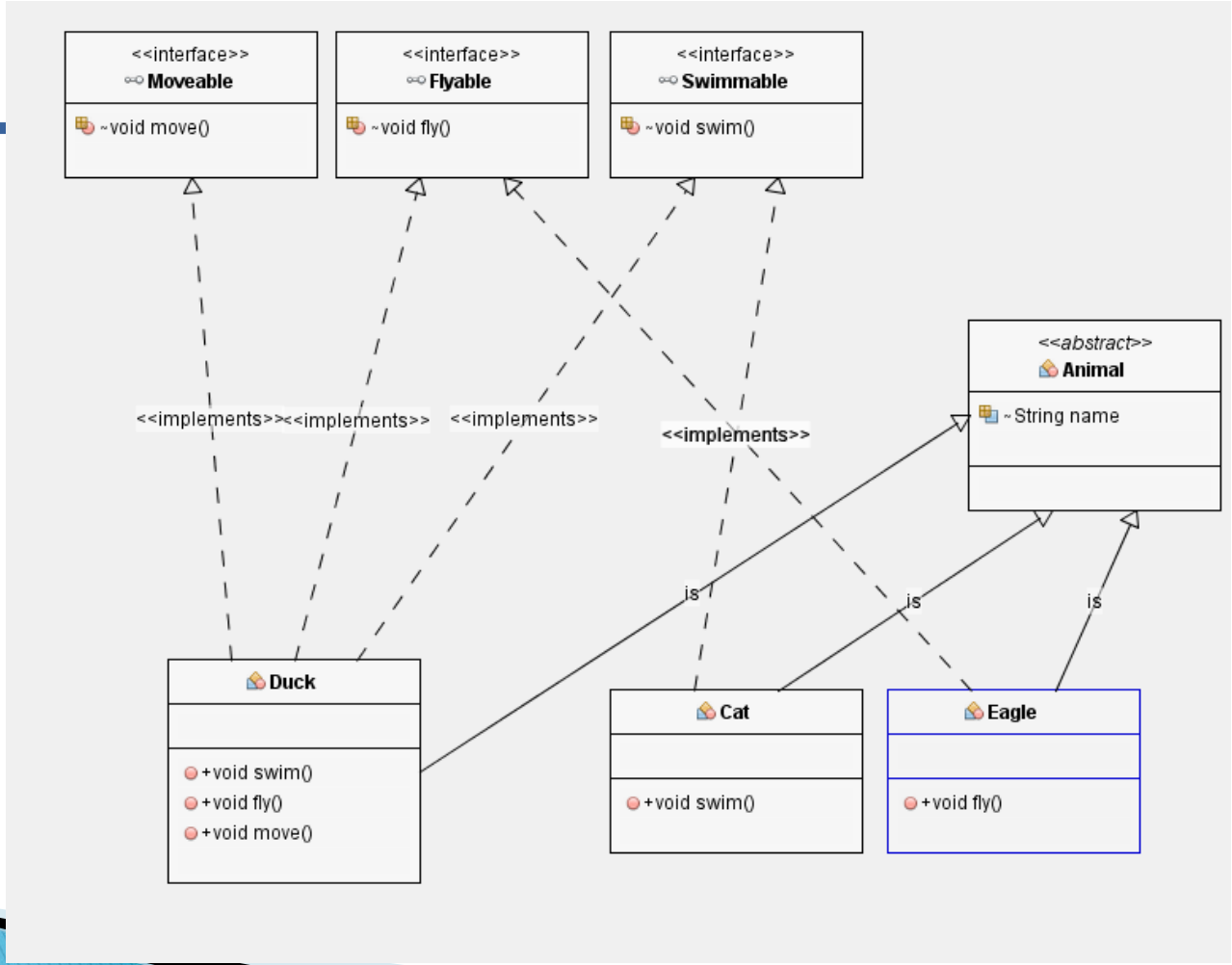
Abstraction using Abstract classes and Interfaces

Multiple inheritance

- A Classes can inherit multiple interfaces.
- The advantage of multiple inheritance is that it allows a class to inherit the functionality of more than one base class
  - thus allowing for modeling of complex relationships.

```
public interface A {
void a();
}
public interface B {
 void b();
}
public class Test implements A, B {
  public void a() {}
  public void b() {}
}
```


Print();
ColoredPrint();
Fax();
Scan();
Printer
Canon Professional
Print(){}
ColoredPrint(){}
Fax(){}
Scan(){}
Canon Basic
Print(){}
Scan(){}
Canon Office
Print(){}
Fax(){}
Scan(){}


<<interface>>
Moveable
~void move()
<<interface>>
Flyable
~void fly()
<<interface>>
Swimmable
~void swim()
<<abstract>>
Animal
~String name
<<implements>>
<<implements>>
<<implements>>
<<implements>>
is
is
is
Duck
+void swim()
+void fly()
+void move()
Cat
+void swim()
Eagle
+void fly()

Instance of

- The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).
- It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

```
class Simple1{
 public static void main(String args[]){
     Simple1 s=new Simple1();
     System.out.println(s instanceof Simple1);//true

 }
}
```

Loosely coupling?

- The Hat is "loosely coupled" to the body.
  - This means you can easily take the hat off without making any changes to the person/body.
  - When you can do that then you have *"loose coupling".*
- **In short, loosely coupling makes code easier to change.**

if you want to change the skin, you would also HAVE TO change the design of your body as well because the two are joined – **they are** *"tightly coupled"*

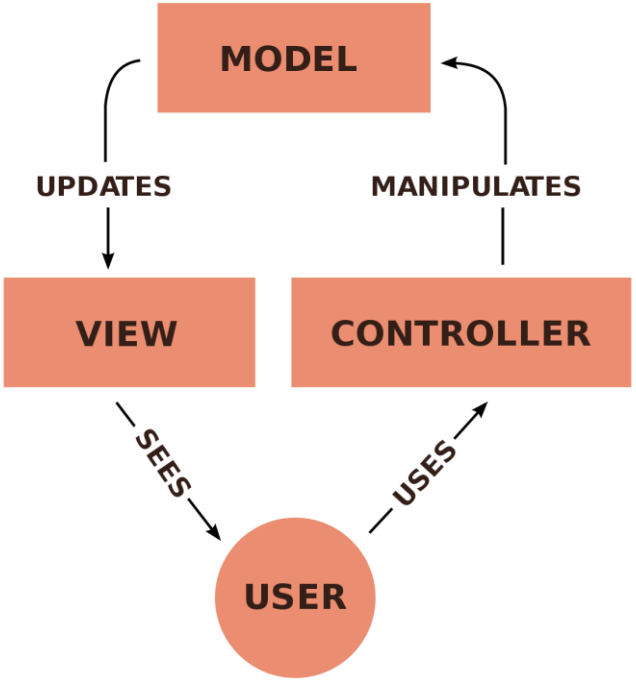
The Hat is "loosely coupled“ to the body.

loose coupling means that you need to know less of the used code. Thus using interfaces provides a part of that.

MVC is a loosely coupled

Loose coupling is simply writing software in such a way that all your classes can work independently without relying on each other. MVC is a loosely coupled

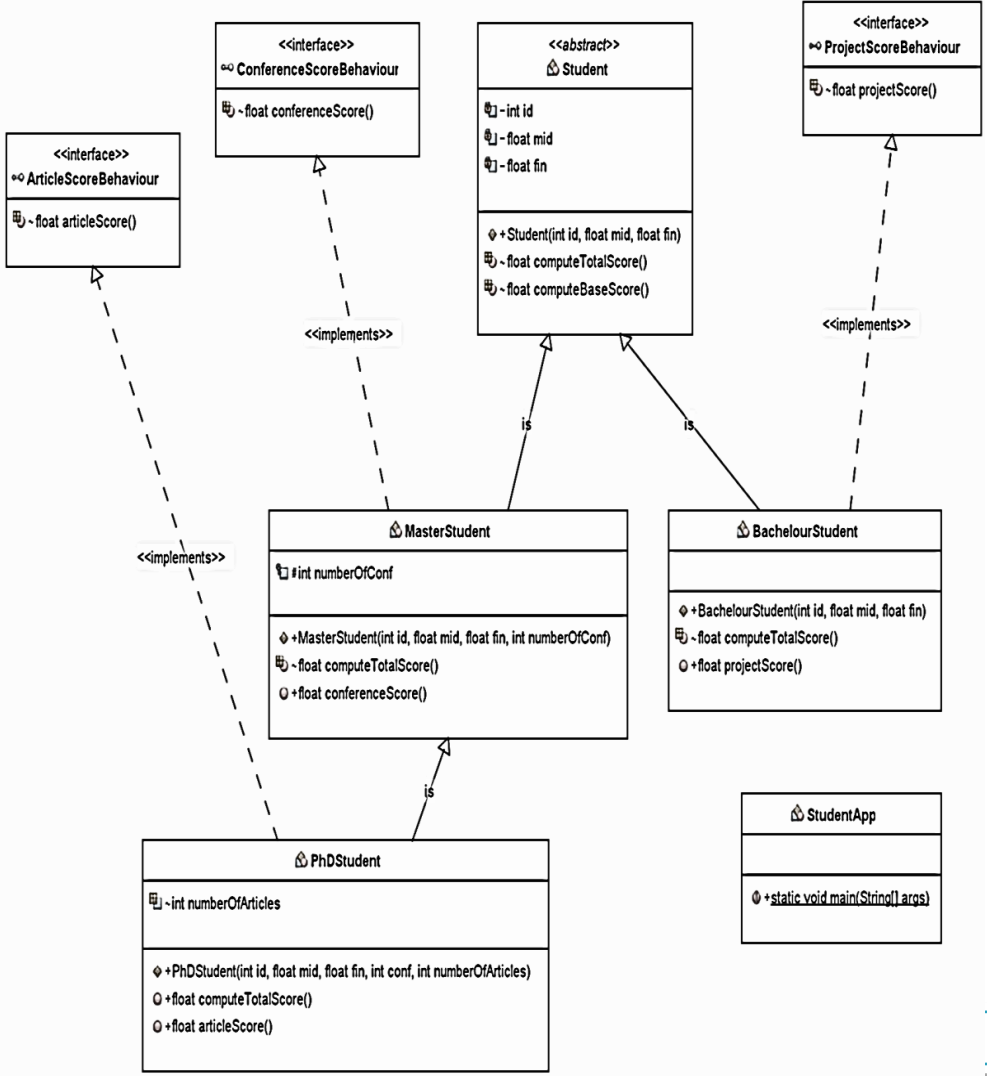
- Model - Contains your classes and business logic
- View - Displays the application UI based on the model data
- Controller - Respond to an oncoming URL request and select the appropriate view



Lab exercise

Write the classes and interfaces given by following diagram. Override methods for each class given by table (symbols: #protected, ~ default -private, + public).

Note: Use the table in the next slide.



	computeTotalScore()	computeBaseScore()	projectScore()	conferenceScore()	articleScore()
Student	*abstract*	Mid * 0.4 + fin * 0.6	--	--	--
BachelourStudent	computeBaseScore() + projectScore()	--	20	--	--
MasterStudent	computeBaseScore() + conferenceScore()	--	--	numberOfConf * 5	--
PhDStudent	computeTotalScore() + articleScore()	--	--	--	numberOfArticle*8

Thanks ☺