ALGONQUIN COLLEGE

CST8132 OBJECT ORIENTED PROGRAMMING

Properties of OOP

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Professor : Dr. Anu Thomas

Email: thomasa@algonquincollege.com

Office: T314

Today's Topics

- Lab 2 – Store Management System I
- Features of OOP
  - Encapsulation
  - Polymorphism
  - Abstraction
  - Inheritance

Properties of Object Oriented Programming

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Abstraction

- Process of finding commonalities between different objects
  - Results in hierarchy of superclass-subclass
    - Inheritance
  - Also, defining an abstract behavior to represent common behavior of subclasses
    - Subclasses may implement the behavior in different ways
      - polymorphism

Examples of Objects

Surgeon

1. Emp ID
2. Name
3. Address
4. Telephone
5. Base salary
6. Specialization
7. Hospital
8. Number of Surgeries

Family Doctor

1. Emp ID
2. Name
3. Address
4. Telephone
5. Base salary
6. Clinic name
7. Number of days working in the clinic

What do you see here?

Examples of Objects - Composition

Person
1. Name
2. Address
3. Telephone

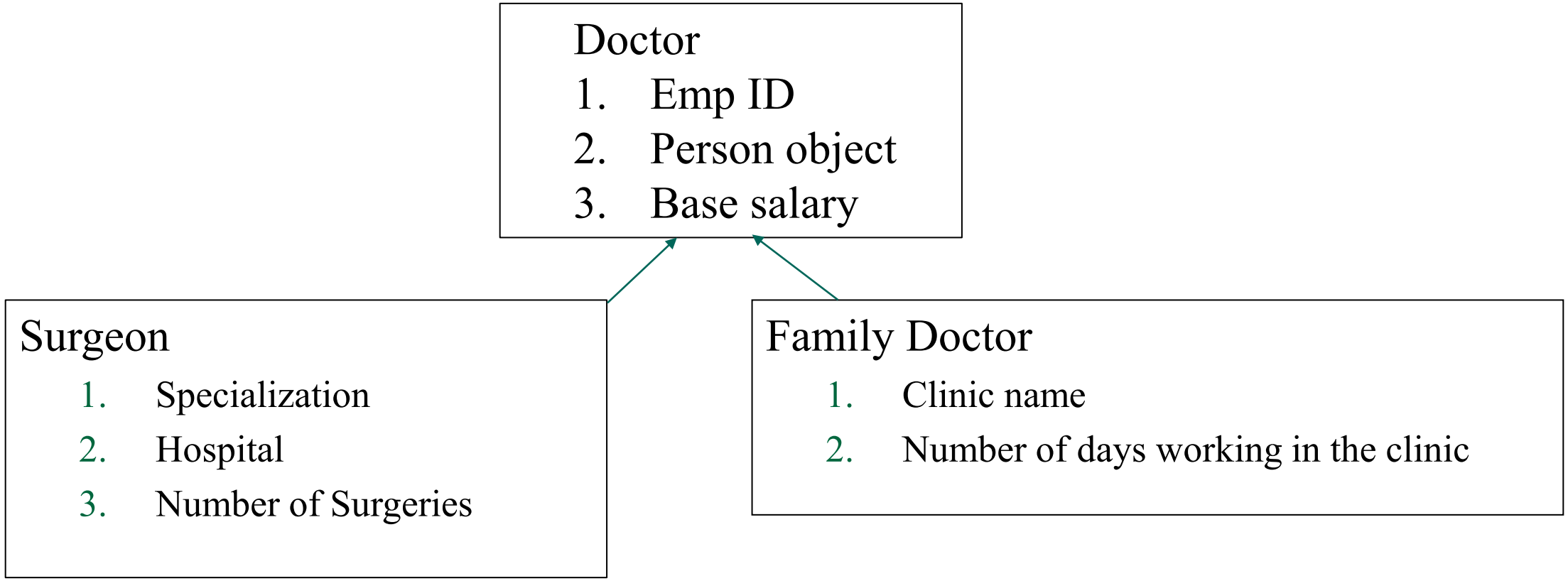
Surgeon
1. Emp ID
2. Person object
3. Base salary
4. Specialization
5. Hospital
6. Number of Surgeries

Family Doctor
1. Emp ID
2. Person object
3. Base salary
4. Clinic name
5. Number of days working in the clinic

What do you see here?

Surgeon & Family Doctor “has” personal properties - Composition

Examples of Objects - Inheritance



What do you see here? Inheritance

Surgeon "is a" Doctor vs Surgeon has a Doctor(not true)

Family Doctor "is a" Doctor

Example

- Surgeon and Family Doctor share attributes
  - Surgeon and Family Doctor inherit from Doctor
  - Doctor is called the super-class
  - Surgeon and Family Doctor are called sub-classes

Inheritance

- When we are writing our classes, we can use an existing class as a starting point
- Suppose we have a `Doctor` class, and we want to add a `FamilyDoctor` class that is based on `Doctor`
- We would say `FamilyDoctor` "is-a" `Doctor`
- `FamilyDoctor` is a new class based on `Doctor`
- `FamilyDoctor` will inherit the attributes and methods of `Doctor`
- More `FamilyDoctor`-specific attributes and methods can be added
- Example:

```
public class FamilyDoctor extends Doctor{

}
```

Inheritance (cont'd)

- With inheritance, we can create new classes from existing classes
- extends keyword: `public class B extends A`
- `A` is the superclass
- `B` is the subclass
- `B` inherits all of the members of `A`
  - The members are the fields and methods
  - The constructors of the superclass is invoked by the subclass constructors
  - Private members of `A` are not visible in `B`
  - `B` can add new fields and members
  - `B` can override methods of `A`
  - If a method in `A` named `doIt()` is overridden in `B`, then in `B`, `doIt()` invokes the `B` version, and `super.doIt()` invokes the `A` version.

Access Modifiers

- Private
  - Only instances of the class itself can access it
- Protected
  - Only instances of the class and instances of the subclasses can access
- Public
  - Everyone has access. can be accessed from within the class, outside the class, within the package and outside the package
- Default
  - Members of the same package can access all members

Access modifiers (contd.)

Access Modifier	Java keyword	UML symbol	Within class	Within package	Is subclass	Not subclass
Public	public	+	Y	Y	Y	Y
Protected	protected	#	Y	Y	Y	N
Package		~	Y	Y	N	N
Private	private	-	Y	N	N	N

Java Constructors and Inheritance

- If no constructor is defined for a class, a default constructor will be implicitly defined
- The default constructor for the super-class is automatically implicitly called by the constructor of a sub-class (unless at the beginning, super is called explicitly)
- If any non-default constructor is defined, there is no implicit default constructor which can cause compile time errors when a subclass needs the superclass to have a default constructor
- this() can be used in a constructor to refer to another constructor of the current class.
  - Ex: a constructor `Square()` referring to `Square(int sideLength)` as in:

```
public Square(){
  this(4);      //if the sideLength is not specified, a sideLength of 4 is used
}
```

Inheritance and Overriding methods

- When one class extends another class, in the subclass, the programmer can override a method with the same signature in the superclass
- We use the @Override annotation on the method in the subclass
- The subclass version of the method will be the version used by the objects of the subclass
- If we want to use the superclass version, we can use

```
super.methodName();
```

Inheritance and Polymorphism

- Shapes Example
  - A circle “is-a” shape
  - A triangle “is-a” shape
  - We will make shape a parent class
  - Demonstration of calculating areas of different shapes
- Bicycles example of polymorphism
  - http://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html

Example – Shape, Rectangle & Square classes

```java
public class Shape {

    protected double area;
    protected double perimeter;

    public void findArea() {
        area = 0;
    }

    public void findPerimeter() {
        perimeter = 0;
    }

    public void printArea() {
        System.out.println("Area is " + area);
    }

    public void printPerimter() {
        System.out.println("Perimter is " + perimeter);
    }
}
```

```java
public class Square extends Shape{
    private double side;

    Square(){}

    Square(int s){
        side = s;
    }
    @Override
    public void findArea() {
        area = side *side;
    }

    @Override
    public void findPerimeter() {
        perimeter = 4 * side;
    }
}
```

```java
public class Rectangle extends Shape{

    private double length;
    private double width;

    Rectangle(){
    }

    Rectangle(double l, double w){
        length = l;
        width = w;
    }

    @Override
    public void findArea() {
        area = length * width;
    }

    @Override
    public void findPerimeter() {
        perimeter = 2 * (length +width);
    }
}
```

```java
public class ShapeTest {

    public static void main(String[] args) {

        //Shape object can take the form of a Shape, Rectangle or a Square
        Shape s1 = new Shape();

        Shape s2 = new Rectangle(3,4);

        Shape s3 = new Square(5);

        System.out.println("Shape s1 which is a Shape object");
        s1.findArea();
        s1.findPerimeter();
        s1.printArea();
        s1.printPerimter();

        System.out.println("\nShape s2 which is a Rectangle object");
        s2.findArea();
        s2.findPerimeter();
        s2.printArea();
        s2.printPerimter();

        System.out.println("\nShape s3 which is a Square object");
        s3.findArea();
        s3.findPerimeter();
        s3.printArea();
        s3.printPerimter();
    }
}
```

Comments NOT included to save space.

Hospital System

Comments NOT included to save space.

```java
public class Person {
    private String firstName;
    private String lastName;
    private String email;
    private long phone;

    Person() {}

    Person(String fName, String lName, String email, long ph) {
        firstName = fName;
        lastName = lName;
        this.email = email;
        phone = ph;
    }

    public String getName() {
        return firstName + " " + lastName;
    }

    public String getEmail() {
        return email;
    }

    public long getPhone() {
        return phone;
    }
}
```

```java
public class Doctor {
    private int empId;
    private Person p; // Composition- Without this attribute, this class will not exist.
    protected double baseSalary;

    Doctor() {}

    Doctor(int id, String n1, String n2, String e, long ph, double sal) {
        empId = id;
        p = new Person(n1, n2, e, ph);
        baseSalary = sal;
    }

    public void printDoctor() {
        System.out.printf("%6d | %15s | %15s | %12d | ", empId, p.getName(),
                p.getEmail(), p.getPhone());
    }

    public void readDoctor() {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter ID: ");
        empId = input.nextInt();
        System.out.print("Enter first Name: ");
        String fName = input.next();
        System.out.print("Enter last Name: ");
        String lName = input.next();
        System.out.print("Enter email: ");
        String email = input.next();
        System.out.print("Enter phone: ");
        long ph = input.nextLong();
        p = new Person(fName, lName, email, ph);
        System.out.print("Enter salary: ");
        baseSalary = input.nextDouble();
    }
}
```

```java
public class Surgeon extends Doctor{

    private String specialization;
    private String hospital;
    private int numSurgeries;

    @Override
    public void readDoctor() {
        Scanner input = new Scanner(System.in);
        super.readDoctor();
        System.out.print("Enter hospital name: ");
        hospital = input.nextLine();
        System.out.print("Enter specialization: ");
        specialization = input.next();
        System.out.print("Enter number of surgeries: ");
        numSurgeries = input.nextInt();
    }

    @Override
    public void printDoctor() {
        super.printDoctor();
        System.out.printf("%12s | %12s | %8.2f |\n", hospital, specialization, baseSalary + numSurgeries*1000);
    }
}
```

```java
public class FamilyDoctor extends Doctor{
    private String clinicName;
    private int numDays;

    @Override
    public void readDoctor() {
        Scanner input = new Scanner(System.in);
        super.readDoctor();
        System.out.print("Enter clinic name: ");
        clinicName = input.next();
        System.out.print("Enter number of days working in this clinic: ");
        numDays = input.nextInt();
    }

    @Override
    public void printDoctor() {
        super.printDoctor();
        System.out.printf("%12s | %8.2f |\n", clinicName, baseSalary + numDays * 4 * 12 * 100);
    }
}
```

Hospital System (Contd.)

```java
public class Hospital {

    private Doctor []doctors;

    Hospital(){}

    Hospital(int n){
        doctors = new Doctor[n];
    }

    public void readDoctors() {
        Scanner input = new Scanner(System.in);
        for(int i=0; i<doctors.length; i++) {
            System.out.print("1. Surgeon \n2. Family Doctor \nEnter Doctor's type:");
            int type = input.nextInt();
            if(type == 1)
                doctors[i] = new Surgeon();
            else if (type == 2)
                doctors[i] = new FamilyDoctor();

            doctors[i].readDoctor();
        }
    }

    public void printDoctors() {
        for(int i=0; i<doctors.length; i++)
            doctors[i].printDoctor();
    }
}
```

```java
public class HospitalTest {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.print("How many doctors do you want to add to the system: ");
        int num = input.nextInt();

        Hospital h = new Hospital(num);

        h.readDoctors();
        h.printDoctors();

    }
}
```

`static` keyword

- Discussion of `static` versus instance
  - Static variables and static methods are a part of the class itself
  - All objects of the class, if any, use the SAME copy of them
  - Static variables and methods can be used BEFORE any objects of that class are instantiated
  - Static members should be referenced by the `ClassName` itself
  - Example of using static method “sort” of Arrays class:
    - `Arrays.sort(myArray)`

    (reference for Arrays class: https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html)
  - Instance members do not exist until an object has been created (with the **new** keyword)
  - Instance members are accessed through a reference to the object

In-class Exercise – College System

- Create different classes and think about their relationships
- Student (Full time student, Part time Student), Professor (Full time Professor, Part time Professor)
- Student and Professor “has” personal properties
- Full time Student “is a” Student
- Part time Student “is a” Student
- Full time Professor “is a” Professor
- Part time Professor “is a” Professor