Advanced Topics in Programming

LAB 2 – CLASSES, INHERITANCE, INTERFACE

Object Orientation in Java

Recap

- Java is an OOP language (almost pure)
- OOP languages uses
  - Objects as data structures (fields and methods).
  - Data abstraction, encapsulation, modularity, polymorphism and inheritance.
- An object is an instance of class.
- A class is loaded only once while its instances can be as many as we whish.

Recap

- An instance is created with “new” command.
- “new” Calls the class’s constructor.
- Members are called from objects.
- Static members are called from the class.

```java
public class HelloWorld {
 public void print() {
   System.out.println("Hello World!");
 }
}
```

```java
public class Run {
 public static void main(String[] args) {
  HelloWorld h=new HelloWorld();
  h.print();
 }
}
```

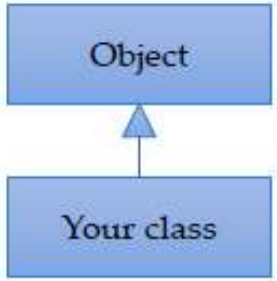
Recap

- Given a class A with static methods s() and a method m()
- We define A a;

- What of the following will work?
  - a.s(); ✗
  - a.m(); ✗
  - A.s(); ✓
  - A.m(); ✗

Object Class

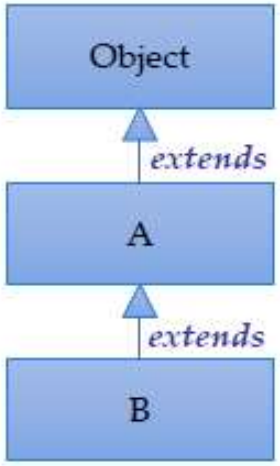
- Every class in Java inherited the class Object.
- Object is the most general class.
- Used for general purpose, e.g.,
  - Method(Object arg0) – arg0 can be any object.
  - Object array[] – can store any objects.
- Object's methods:

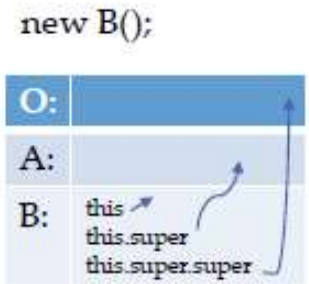



```
Object o=new Object();
o.
    equals(Object obj) : boolean - Object
    getClass() : Class<?> - Object
    hashCode() : int - Object
    notify() : void - Object
    notifyAll() : void - Object
    toString() : String - Object
    wait() : void - Object
    wait(long timeout) : void - Object
    wait(long timeout, int nanos) : void - Object
```

Inheritance

- In order to avoid the diamond of death Java allows a class to inherit only 1 Class.
- Inheritance keyword is extend.
- A sub-class extends a super-class.
- this – refers to the current instance of a class.
- super – refers to the super class's instance.





Abstract classes

- Until now we've seen what can and cannot be extended.
- But what if we want to force an implementation of subclass?
- When abstract is attached to a method:
  - It is left unimplemented.
  - The class becomes abstract, and cannot be instanced.
  - Only a subclass that implemented the abstract method can be instanced.

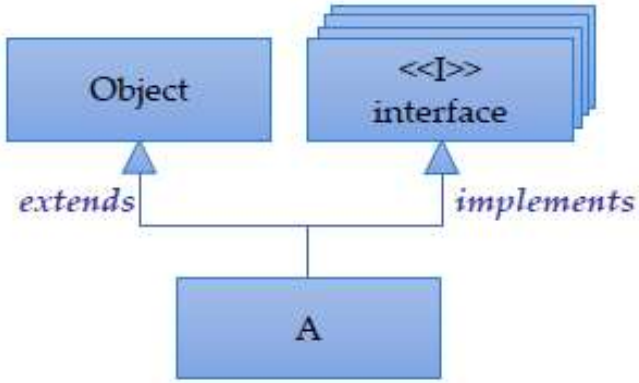
Abstract classes

- A concrete subclass can be instanced
  - Inherit all implemented methods as usual
  - But must implement all abstract methods

```
public abstract class MyAbstract {
 String name;
 public MyAbstract(String name){
  this.name=name;
 }
 public abstract void welcome();
}
```

```
public class MyConcrete extends MyAbstract{
 public MyConcrete(String name) {
  super(name);
 }
 @Override
 public void welcome() {
  System.out.println("wlcome "+name);
 }
}
```

Interfaces



- Interfaces are pure abstract classes
- Defined by the keyword interfsce
- Interfaces are implemented (not extended)
  - By the keyword is implements
- Multiple implementation of interfaces is allowed in java
- Interfaces are the common language in which objects interact

Interfaces

```java
public interface GuitarPlayer {
    public void playGuitar();
    public void stop();
    //...
}
```

```java
public interface Lecturer {
    public void startTeaching();
    public void checkExams();
    //...
}
```

```java
public class MusicLecturer implements GuitarPlayer, Lecturer {
        @Override
        public void startTeaching() {
                // TODO Auto-generated method stub
        }
        @Override
        public void checkExams() {
                // TODO Auto-generated method stub
        }
        @Override
        public void playGuitar() {
                // TODO Auto-generated method stub
        }
        @Override
        public void stop() {
                // TODO Auto-generated method stub
        }
}
```

```java
// we can't instantiate an interface
//GuitarPlayer guitarPlayer=new GuitarPlayer();
// this is OK:
GuitarPlayer guitarPlayer = new MusicLecturer();
guitarPlayer.playGuitar();
// without casting, we can only expect
// the functionality of a guitar player
//guitarPlayer.startTeaching();
// this is OK:
Lecturer lecturer=new MusicLecturer();
lecturer.startTeaching();
```

UML Class Diagram To Code

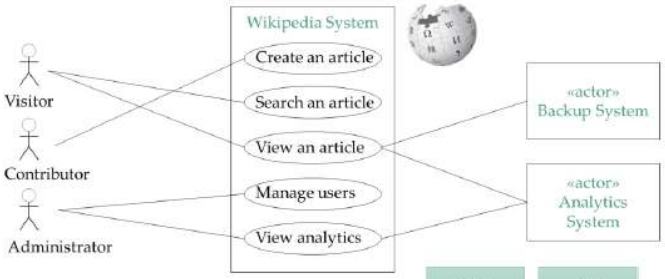
Analysis – Understand our problem & required functionality

Object Oriented Design – Plan a solution that meets these requirements

Programming – Implement the design with an OOP

Object Oriented Analysis – the steps

1. Gather **requirements**

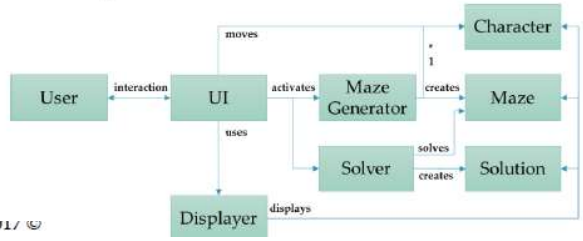


2. **Describe** the application

3. **Identify** the main objects

4. Describe the **interaction** between these objects

5. Create a **class diagram**



Dept. of Software and Information Systems Engineering

Dr. Eliahu Khalastchi, 2017 ©

UML

UML

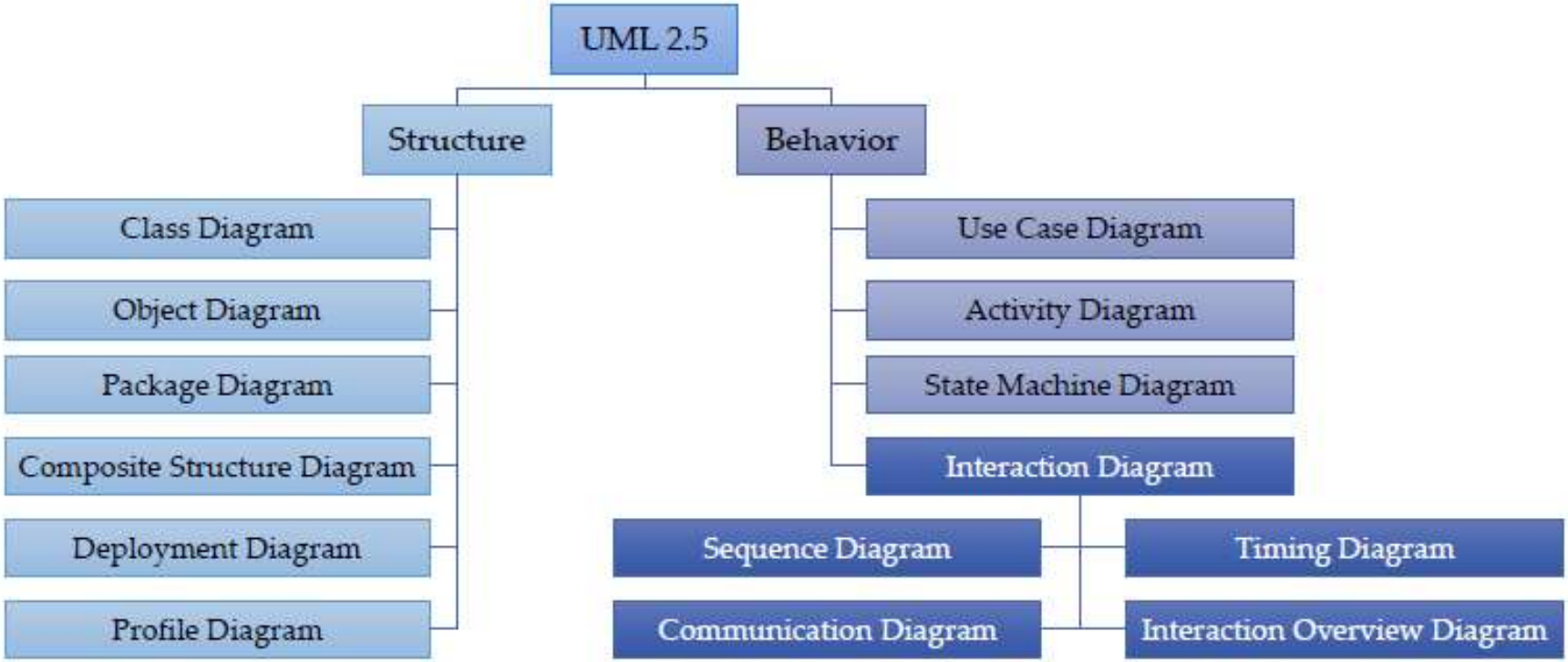
Unified Modeling Language

UNIFIED
MODELING
LANGUAGE
™

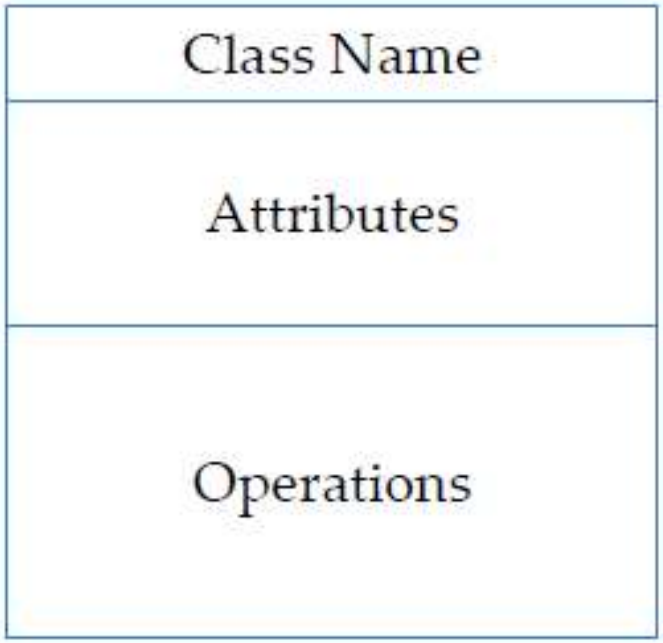
Classification of UML Diagrams

- UML specification defines two major kinds of UML diagrams:
  - Structure diagrams
    - Show the static structure of the system
    - Its parts on different abstraction and implementation levels
    - How they are related to each other
  - Behavior diagrams
    - show the dynamic behavior of the objects in the system
    - Which can be described as a series of changes to the system over time

Classification of UML Diagrams

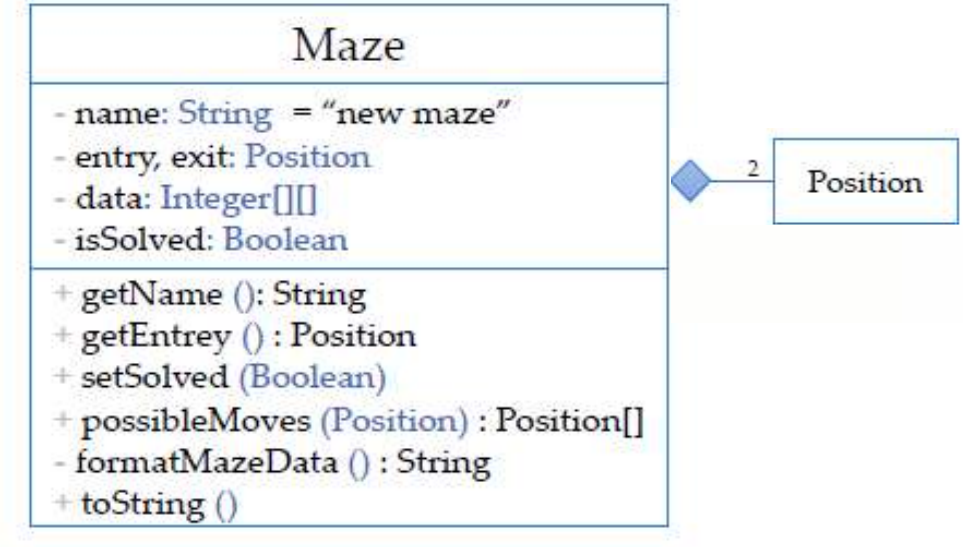


Class Diagram



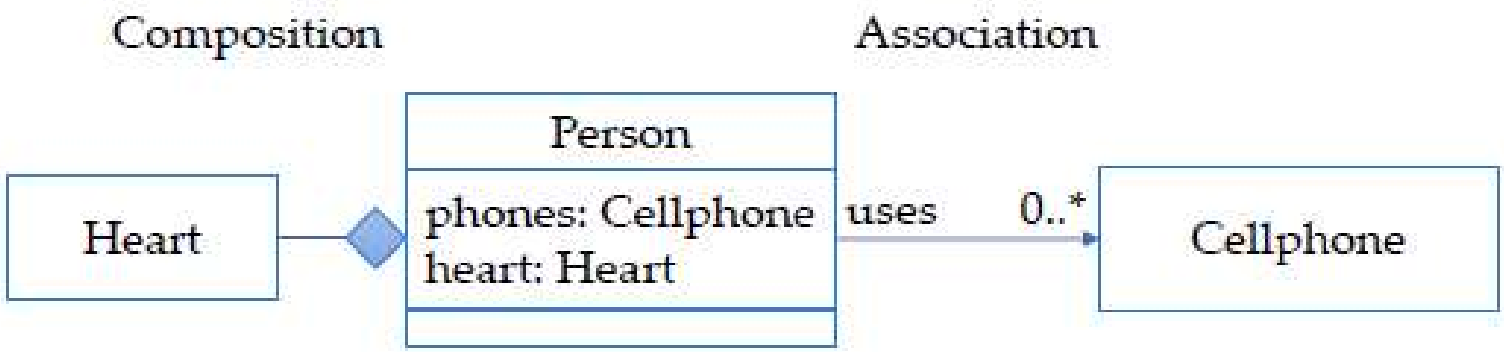
Class Diagram – a class

+	Public
-	Private
#	Protected
/	Derived
~	Package

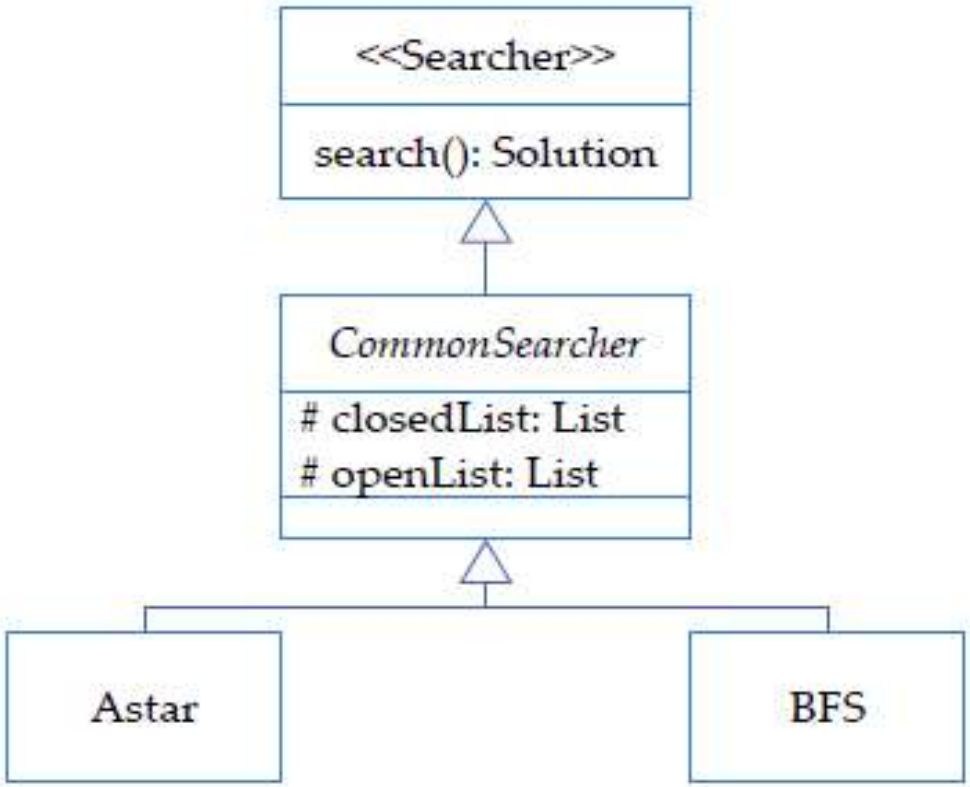


Class Diagram - association

Class Diagram - association

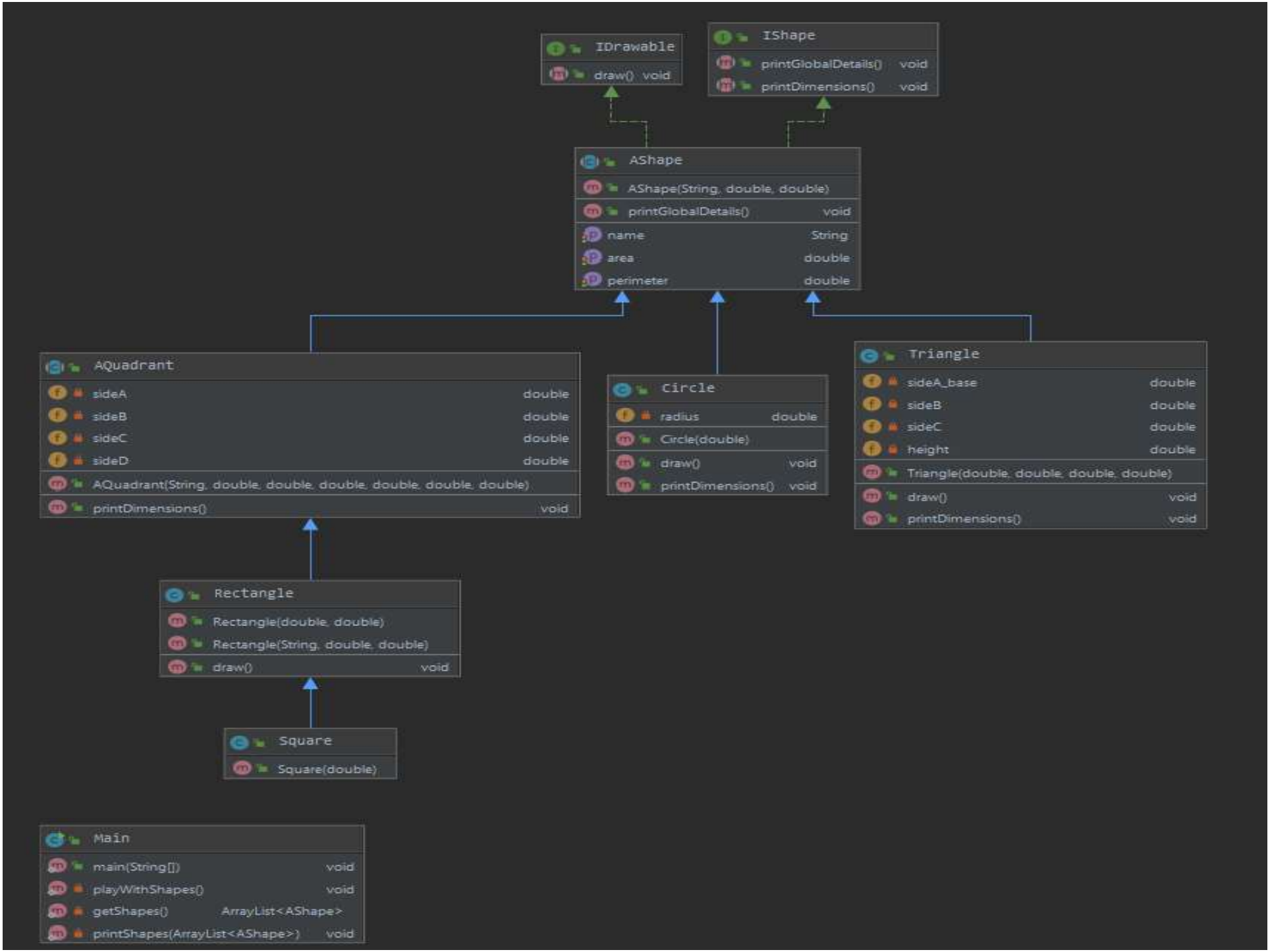


Class Diagram – Generaliztion



Lab Exercise

From UML class diagram to Code


IDrawable
draw() void
IShape
printGlobalDetails() void
printDimensions() void
AShape
AShape(String, double, double)
printGlobalDetails() void
name String
area double
perimeter double
AQuadrant
sideA double
sideB double
sideC double
sideD double
AQuadrant(String, double, double, double, double, double, double)
printDimensions() void
Circle
radius double
Circle(double)
draw() void
printDimensions() void
Triangle
sideA_base double
sideB double
sideC double
height double
Triangle(double, double, double, double)
draw() void
printDimensions() void
Rectangle
Rectangle(double, double)
Rectangle(String, double, double)
draw() void
Square
Square(double)
Main
main(String[]) void
playWithShapes() void
getShapes() ArrayList<AShape>
printShapes(ArrayList<AShape>) void