# Abstract Class

COMP2026

PROBLEM SOLVING USING OBJECT ORIENTED PROGRAMMING

## Consider the following inheritance hierarchy

```java
public class Shape
{
    private String name;
    public Shape(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
}
```

```java
public class Square extends Shape
{
    private double side;
    public Square(String name, double side){
        super(name);
        this.side = side;
    }
    public double getSide()
    {
        return this.side;
    }
    public void setSide(double side) {
          this.side = side;
    }
}
```

```java
public class Circle extends Shape
{
    private double radius;
    public Circle(String name, double radius){
        super(name);
        this.radius = radius;
    }
}
```

## Abstract Class

- Inheritance makes code reuse more systematic and easier to maintain
- However, sometimes a superclass can be **too general**, which we are very **unlikely to use it for object instantiation**. It only acts as a "Framework"
- We can define such classes as **abstract classes**

# Consider the following inheritance hierarchy

```
public abstract class Shape
{
    private String name;
    public Shape(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
}
```

Shape can be defined as abstract class as it is not likely to create a shape object

```
public class Square extends Shape
{
    private double side;
    public Square(String name, double side){
        super(name);
        this.side = side;
    }
    public double getSide()
    {
        return this.side;
    }
    public void setSide(double side) {
          this.side = side;
    }
}
```

```
public class Circle extends Shape
{
    private double radius;
    public Circle(String name, double radius){
        super(name);
        this.radius = radius;
    }
}
```

# Abstract Class

- We cannot use new operator to create objects of the Shape class

```
Shape s;
s = new Shape(...);  //Error!
```

- But we could create Shape variables

```
Shape s;
s = new Circle(...);  //OK, Circle object
```

## Abstract Methods

- Abstract methods can be declared in abstract classes. They are **methods declared without any implementation** (no method body), like this:

```
public abstract double area();
```

- The actual implementation of the abstract method should be written in the subclass. Different subclasses can have different logic of the same abstract method

# Consider the following inheritance hierarchy

```
public abstract class Shape
{
     private String name;
     public Shape(String name){
          this.name = name;
     }
     public String getName(){
          return name;
     }
     public abstract double area();
     public abstract double perimeter();
}
```

Subclass has to provide implementations for ALL of the abstract methods in its parent class, otherwise it has to be declared abstract

```
public class Square extends Shape
{
     private double side;
     public Square(String name, double
                          side){
          super(name);
          this.side = side;
     }
     ...
     public double area(){
          return side * side;
     }
     public double perimeter(){
          return side * 4;
     }
}
```

```
public abstract class Circle extends Shape
{
     private double radius;
     public Circle(String name, double radius){
          super(name);
          this.radius = radius;
     }
}
```

## Important Notes

- An abstract class **may or may not** include abstract methods.
- **Only** abstract classes can contain abstract methods.
- Not all methods in an abstract class have to be abstract, i.e. some methods can be implemented directly in the abstract class, and the subclass inherits them
- Abstract classes **CANNOT** be instantiated.
- Abstract classes can be inherited (In fact, you want it to be inherited)
- If a subclass is inheriting an abstract class, it has to provide implementations for **ALL** of the abstract methods in its parent class; otherwise, it has to be declared abstract.

## Part A
## Discovery Exercises

Type your answer in **XXXXXXXX_lab12.docx**

## Part B
## Programming Exercises

## Hints for Task 2

- How to find Books, Magazines, KidsMagazines in the ArrayList?
  - Use **instanceof**

```
for (Publicatoin p : pList){
    if(p instanceof Book) {
        //do something with books
        ...
    }
}
```

## Lab Exercise Submission

- Submit the following to Moodle
  - ***XXXXXXXX_ lab12.docx***
  - ***XXXXXXXX_lab12.zip***

*Replace "***XXXXXXXX***" with your **student ID**

**Deadline: Before next Monday noon**

## References

- Dean, J., & Dean, R. (2008). Introduction to programming with Java: A problem solving approach. Boston: McGraw-Hill.
- Forouzan, B. A., & Gilberg, R. F. (2007). Computer science: A structured programming approach using C (3rd ed.). Boston, MA: Thomson Course Technology.
- Gaddis, T. (2016). Starting out with Java (6th ed.). Pearson.
- Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8th ed.). Pearson.
- Schildt, H. (2006). *Java a beginner's guide*. New York: McGraw Hill.
- Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education
- Xavier, C. (2011). Java programming: A practical approach. New Delhi: Tata McGraw Hill.
- Zakhour, S., Kannan, S., & Gallardo, R. (2013). The Java tutorial: A short course on the basics (5th ed.).
- yet another insignificant Programming Notes. (n.d.). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming