ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Object Oriented Programming

## Lecture 2: Concepts

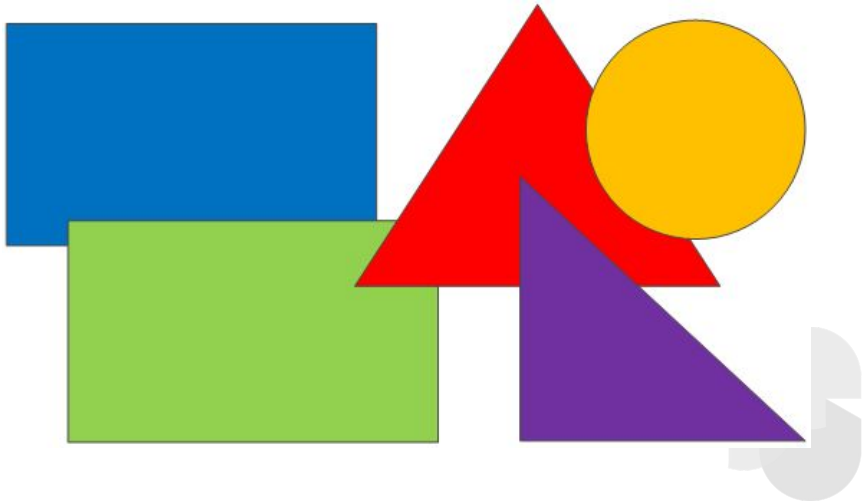

© Spring 2024 - **Prof. Mohamed Sobh**

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Basic Concepts

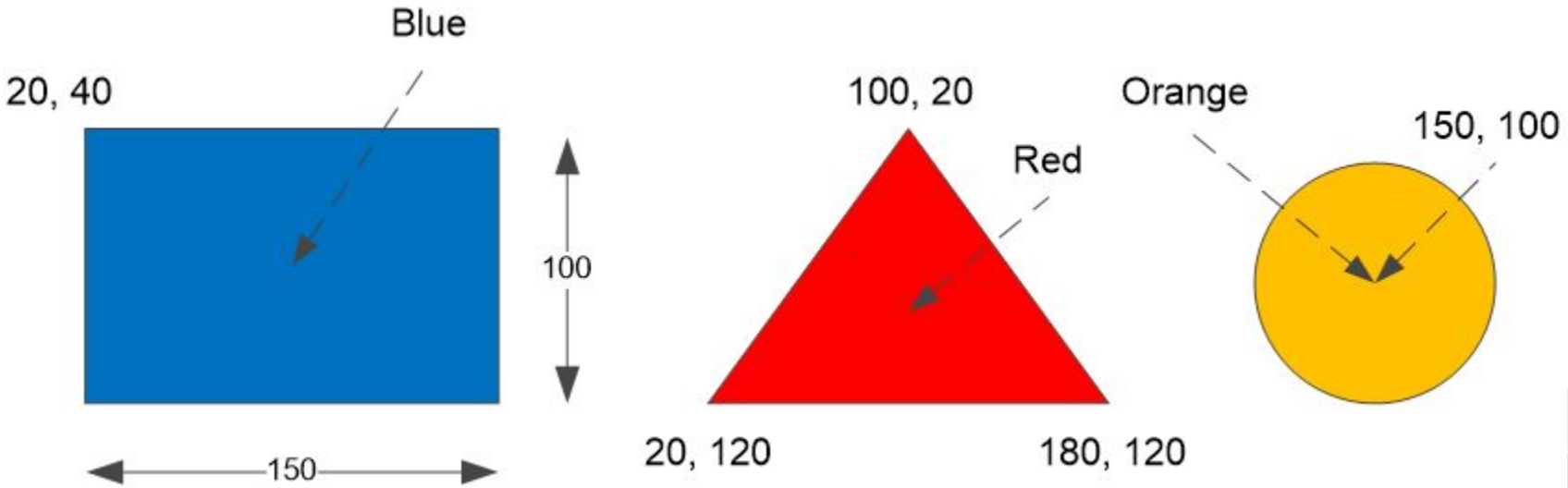
# Objects

- Any system consists of several objects
- A drawing system may consists of
  - Rectangles Objects
  - Triangle Objects
  - Circle Objects



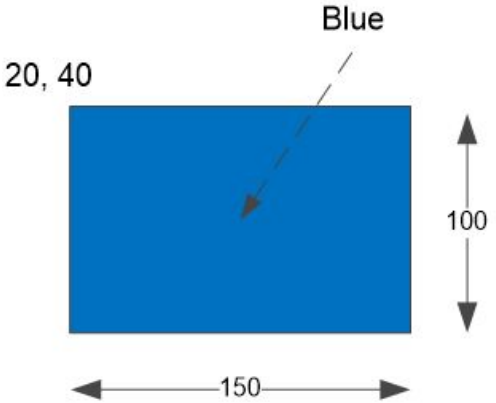
# Objects

- Object has properties (variables, constants)
- Object can do things (methods)



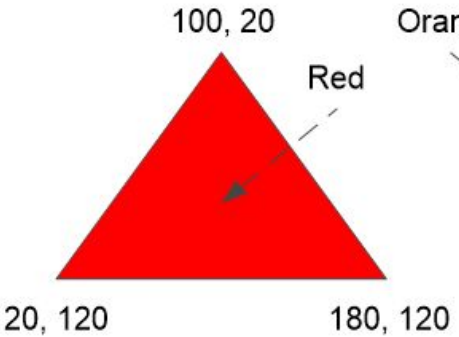
# Classes

Abstraction of objects with same attributes and methods



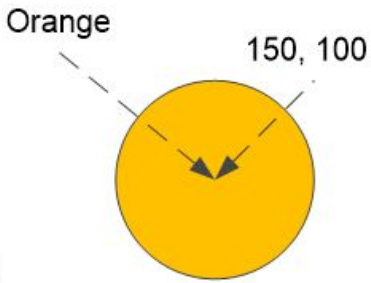

Variables:
X, Y
Width, Height
Color

Methods:
Initialize(X,Y,Width,Height,Color)
Draw (Screen)
Move (dx, dy)
Change Color (Color)
Change Size (ZoomFactor)




Variables:
X1, Y1, X2, Y2, X3, Y3
Color

Methods:
Initialize(X1..Y3, Color)
Draw (Screen)
Move (dx, dy)
Change Color (Color)
Change Size (ZoomFactor)




Constant:
PI 3.143

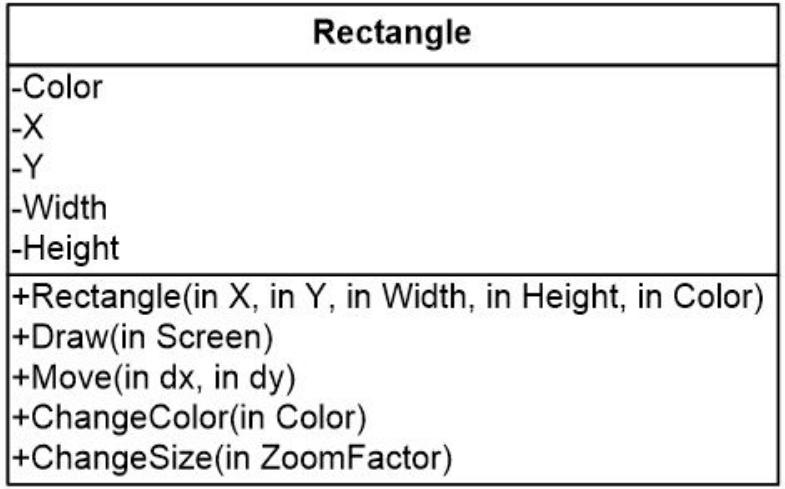
Variables:
X, Y, Radius
Color

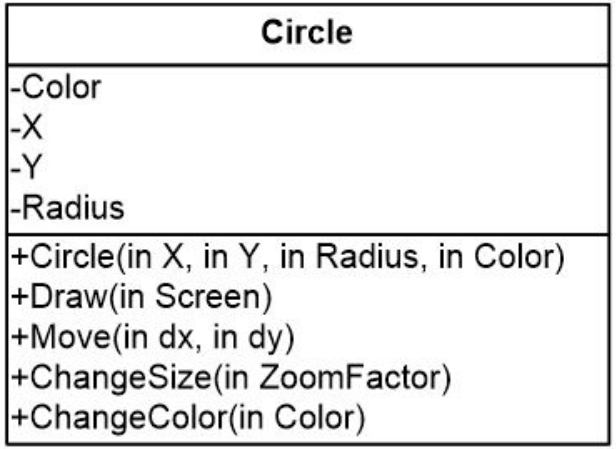
Methods:
Initialize(X, Y, Radius, Color)
Draw (Screen)
Move (dx, dy)
Change Color (Color)
Change Size (ZoomFactor)

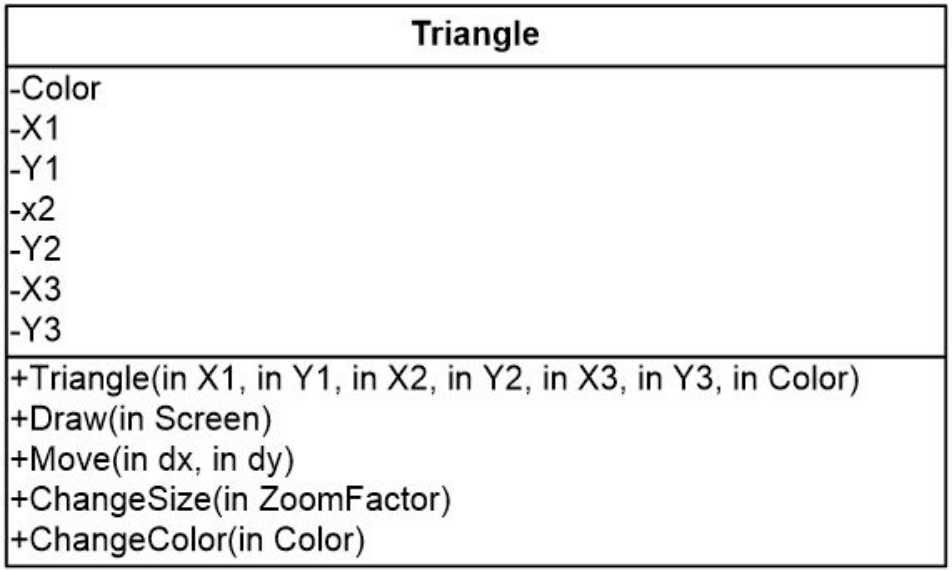
# Classes Properties

- Variables
- Constant and its values
- General Methods
- Initialization Methods
- Un-initialization Method

# UML Classes Definition



# Encapsulation

- Encapsulate Class attributes.
- No way to access attributes directly.
- Only class methods can access the attributes

# Attributes and Methods Encapsulation

- Attributes Encapsulation
  - Public (R/W Externally)
  - Private (-/- Externally)
- Methods Encapsulation
  - Public (Accessible Externally)
  - Private (Inaccessible Externally)

# Rectangle Example

```
public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

# Rectangle Example

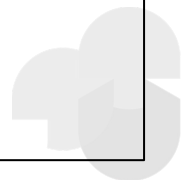
```
public class Rectangle {
    …
    public void print() {
        System.out.printf("x = %d, y = %d, width = %d, height = %d\n",
        x, y, width, height);
    }
    …
}
```

```
Rectangle a = new Rectangle(10, 10, 100, 50);
Rectangle b = new Rectangle(20, 80, 150, 70);
a.print();
b.print();
```

# Rectangle Example

```
public class Rectangle {
    ...
    public int area() {
        return width * height;
    }
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
    ...
}
```

```
Rectangle a = new Rectangle(10, 10, 100, 50);
System.out.println("Area: " + a.area());
a.print();
a.move(20, 20);
a.print();
```



# Employee Example

```
public class Employee {
    private String name;
    private int age;
    private double salary;
    public Employee(String name, int age, double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    public String card() {
        return String.format("Name: %s\nAge:%d\nSalary:%5.2f", name, age, salary);
    }
}
```

# Using Employee Class

```
public class App {
    public static void main(String[] args) throws Exception {
        Employee a = new Employee("Ahmed", 30, 5000.50);
        System.out.println(a.card());
    }
}
```

# Employee Example: increasing employee salary

```
public class Employee {
    ……
    public void addAmountToSalary(int amount) {
        salary += amount;
    }
    public void addPercentageToSalary(double percentage) {
        salary *= 1 + percentage/100.0;
    }
}
```

# Employee Example: increasing employee salary

```
Employee a = new Employee("Ahmed", 30, 5000.50);
System.out.println(a.card());
a.addAmountToSalary(1000);
System.out.println(a.card());
a.addPercentageToSalary(10);
System.out.println(a.card());
```

```java
public class Die {
    private final int MAX = 6;
    private int faceValue;
    public Die() {
        faceValue = 1;
    }
    public void roll() {
        faceValue = (int) (Math.random() * MAX) + 1;
    }
    public int getFaceValue() {
        return faceValue;
    }
}
```

# Die Example

- Class
- Attributes
- Methods
- Constructor

# Die Example: Using Die Class

```
public class App {
    public static void main(String[] args) throws Exception {
        Die die = new Die();
        die.roll();
        System.out.println("Die Face:" + die.getFaceValue());
        die.roll();
        System.out.println("Die Face:" + die.getFaceValue());
        die.roll();
        System.out.println("Die Face:" + die.getFaceValue());
    }
}
```

# Die Example: Implement Build-In toString method

```
public class Die {
    ...
    public String toString() {
        return "face up is "+ faceValue;
    }
}
```

❑ toString

```
Die a = new Die(), b = new Die();
a.roll(); b.roll();
System.out.println("" + a + " - " + b);
```

# Die Example: Giving Name for Die using Constructor

```
public class Die {
     …
    private String name;
    public Die(String name) {
        faceValue = 1;
        this.name = name;
    }
     …
    public String toString() {
        return name + " face up is "+ faceValue;
    }
}
```

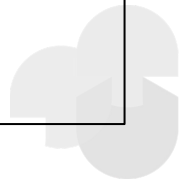
- this

```
Die a = new Die("Blue die");
Die b = new Die("Red die");
a.roll(); b.roll();
System.out.println("" + a + " - " + b);
```

# Die Example: Setting die face value and name

```
public class Die {
    ……
    public void setFace(int faceValue) {
        this.faceValue = faceValue;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

```
Die a = new Die();
a.setFace(5);
a.setName("Blue");
System.out.println(a);
a.roll();
System.out.println(a);
```



# Account Example

```
public class Account {
    private final double RATE = 0.035;
    private long account;
    private double balance;
    private String name;
    public Account(String name, long account, double initial) {
        this.name = name;
        this.account = account;
        this.balance = initial;
    }
}
```

# Account Example

```
public class Account {
    ...
    public double deposit(double amount) {
        balance = balance + amount;
        return balance;
    }
    public double withdraw(double amount, double fee) {
        balance = balance - amount - fee;
        return balance;
    }
    public double addInterest() {
        balance += (balance * RATE);
        return balance;
    }
    public double getBalance() {
        return balance;
    }
    ...
}
```

# Account Example

```
import java.text.NumberFormat;
public class Account {
    ...
    public String toString() {
        NumberFormat fmt = NumberFormat.getCurrencyInstance();
        return (account + "\t" + name + "\t" + fmt.format(balance));
    }
}
```

# Account Example: Using Account Class

```
Account account = new Account("Ted Murphy", 72354, 102.56);
account.deposit(725.85);
System.out.println(account);
account.addInterest();
System.out.println(account);
account.withdraw(500, 50);
System.out.println(account);
```

# Array of Objects

- Create array of objects
- Initialize each with different value
- Iterate over objects

# Grade Example

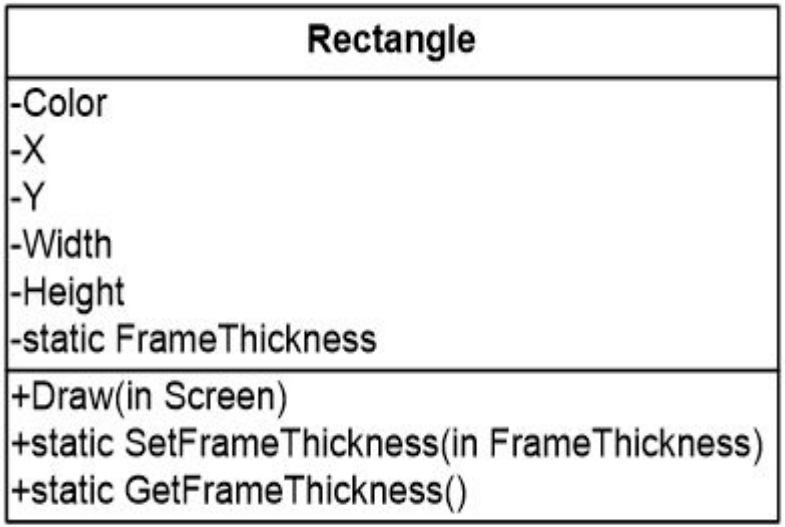
```
public class Grade {
    private int grade;
    public Grade(int grade) {
        this.grade = grade;
    }
    public String toString() {
        if (grade < 50) return "Failed ( " + grade + ")";
        else if (grade < 65) return "Passed ( " + grade + ")";
        else if (grade < 75) return "Good ( " + grade + ")";
        else if (grade < 85) return "Very Good ( " + grade + ")";
        return "Excelent ( " + grade + ")";
    }
}
```

# Grade Example: Using Grade Class

```
Grade[] grades = { new Grade(57), new Grade(86), new
Grade(66), new Grade(95) };

for (Grade grade : grades) {
    System.out.println(grade);
}
```

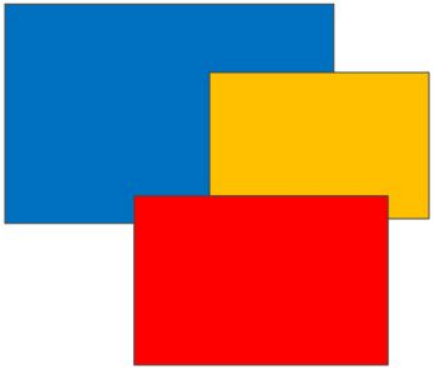
# Static Attributes

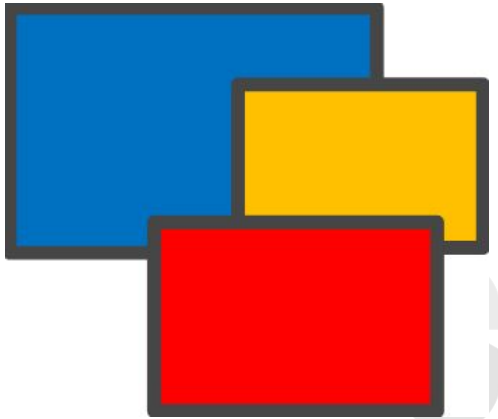
- If it is required to share an information among all objects of the same class, use Static Member
- Static methods are used to access Private Static Attributes



# Static Attributes

| Rectangle |
| --- |
| -Color<br>-X<br>-Y<br>-Width<br>-Height<br>-static FrameThickness = 1 |
| +Draw(in Screen)<br>+static SetFrameThickness(in FrameThickness)<br>+static GetFrameThickness() |





**Rectangle.SetFrameThickness(5)**

# Static Attributes

- Static Attributes share data among multiple objects.
- Static Attributes can be initialized while definition.

# Student Example

```
public class Student {
    private String name;
    private int code;
    private static int codeCounter = 1;
    public Student(String name) {
        this.name = name;
        this.code = codeCounter;
        codeCounter++;
    }
    public String toString() {
        return name + "(" + code + ")";
    }
    public static void setCodeCounter(int codeCounter) {
        Student.codeCounter = codeCounter;
    }
}
```

# Student Example: Using Student Class

```
Student a = new Student("John");
Student b = new Student("Mark");
System.out.println(a);
System.out.println(b);
Student.setCodeCounter(10);
Student c = new Student("Suzan");
System.out.println(c);
```

# Static Methods and Libraries

There is no mean to define several objects from math class.

Because  sin method is static  you can call directly, ex:

double a = math.sin(1.3)

private static attributes and methods can be used internally by public static methods

| **math** |
| --- |
| +final PI = 3.143<br>-final e = 2.7 |
| +static sin(in x)<br>+static cos(in x)<br>+static tan(in x)<br>+static asin(in x)<br>+static acos(in x)<br>+static atan(in x)<br>+static pow(in x)<br>+static log(in x)<br>+static sqrt(in x)<br>-static radtodegree(in x)<br>-static degreetorad(in x) |

# Static Methods and Libraries

- Static methods can be called directly without a need to create an objects
- For that reasons libraries like math consists of static methods performing various mathematical functions

# Simple Method

```
public class MoreMath {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

```
public class App {
    public static void main(String[] args) throws Exception {
        System.out.println(MoreMath.add(5, 6));
    }
}
```

# Passing Array

```
public class MoreMath {
    public static int sum(int[] values) {
        int sum = 0;
        for(int value:values) {
            sum += value;
        }
        return sum;
    }
}
```

```
int[] values = {99, 43, 89, 84, 39, 43};
System.out.println(MoreMath.sum(values));
```

# Passing Object

```
import java.util.ArrayList;
import java.util.Random;
public class MoreMath {
    public static void randFill(ArrayList<Integer> values, int size, int bound)
{
        Random random = new Random();
        for(int i=0;i<size;i++) {
            values.add(random.nextInt(bound));
        }
    }
}
```

```
ArrayList<Integer> values = new ArrayList<Integer>();
MoreMath.randFill(values, 10, 50);
System.out.println(values);
```

# Passing any number of arguments to methods

```
public class MoreMath {
    public static int product(int...values) {
        int product = 1;
        for(int value:values) {
            product *= value;
        }
        return product;
    }
}
```

```
System.out.println(MoreMath.product(7, 4, 8, 7, 4));
```

# Recursive methods

```
public class MoreMath {
    public static int factorial(int x) throws Exception{
        if(x<1) return 1;
        return x * factorial(x-1);
    }
}
```

```
System.out.println(MoreMath.factorial(5));
```

# Throw Exception

```
public class MoreMath {
    public static long factorial(int x) throws Exception{
        if(x<0) throw new Exception("Error: Factorial of -ve not possible...");
        if(x<1) return 1;
        long result = x * factorial(x-1);
        if(result<0) throw new Exception("Error: Factorial is overflow ...");
        return result;
    }
}
```

# Throw Exception

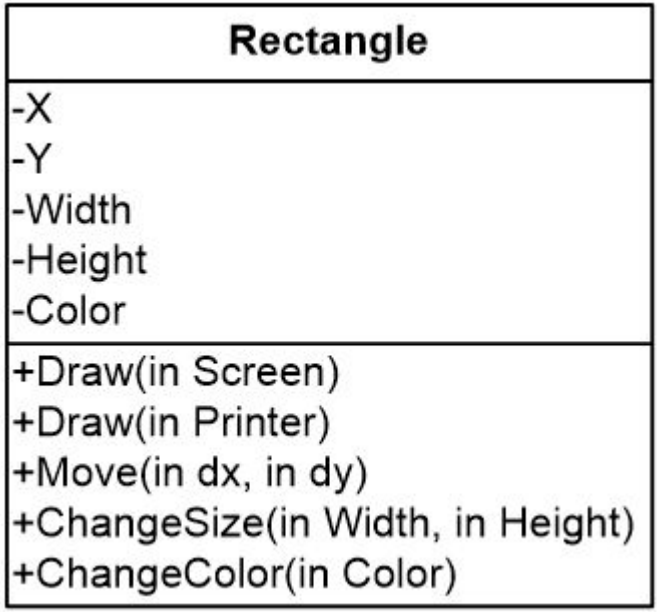
```
while (true) {
    System.out.print("Enter value (Empty to Exit):");
    String text = scanner.nextLine();
    if(text.isEmpty()) break;
    int x = Integer.parseInt(text);
    try {
        long factorial = MoreMath.factorial(x);
        System.out.println(factorial);
    }
    catch(Exception e) {
        System.out.println(e.getMessage());
    }
}
```

# Exercise

- Implement methods
  - sum(num) return sum of values from 1 to num
  - reverse(test) return string in reverse direction (EX: hello → olleh)
- Use
  - Non recursive
  - recursive

# Method Overloading

- Overloaded Methods has the same name with some change in input and operations
- Example: Draw method at Rectangle class can have another version that draw Object to Printer device.



# Method Overloading: Point Class

```
public class Point {
    private int x;
    private int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "(" + x + "," + y + ")";
    }
}
```

# Method Overloading: Rectangle Class

```
public class Rectangle {
    private Point topLeft;
    private Point bottomRight;
    public Rectangle(Point topLeft, Point bottomRight) {
        SetRect(topLeft, bottomRight);
    }
    public Rectangle(int x1, int y1, int x2, int y2) {
        SetRect(x1, y1, x2, y2);
    }
    public void SetRect(Point topLeft, Point bottomRight) {
        this.topLeft = topLeft;
        this.bottomRight = bottomRight;
    }
    public void SetRect(int x1, int y1, int x2, int y2) {
        this.topLeft = new Point(x1, y1);
        this.bottomRight = new Point(x2, y2);
    }
    public String toString() {
        return topLeft + ":" + bottomRight;
    }
}
```

# Method Overloading

```
Rectangle R1 = new Rectangle(10, 10, 200, 150);
Rectangle R2 = new Rectangle(new Point(40, 50), new Point(300, 450));
System.out.println("R1:" + R1 + "\tR2:" + R2);
R1.SetRect(new Point(15, 15), new Point(50, 50));
R2.SetRect(100, 100, 300, 400);
System.out.println("R1:" + R1 + "\tR2:" + R2);
```

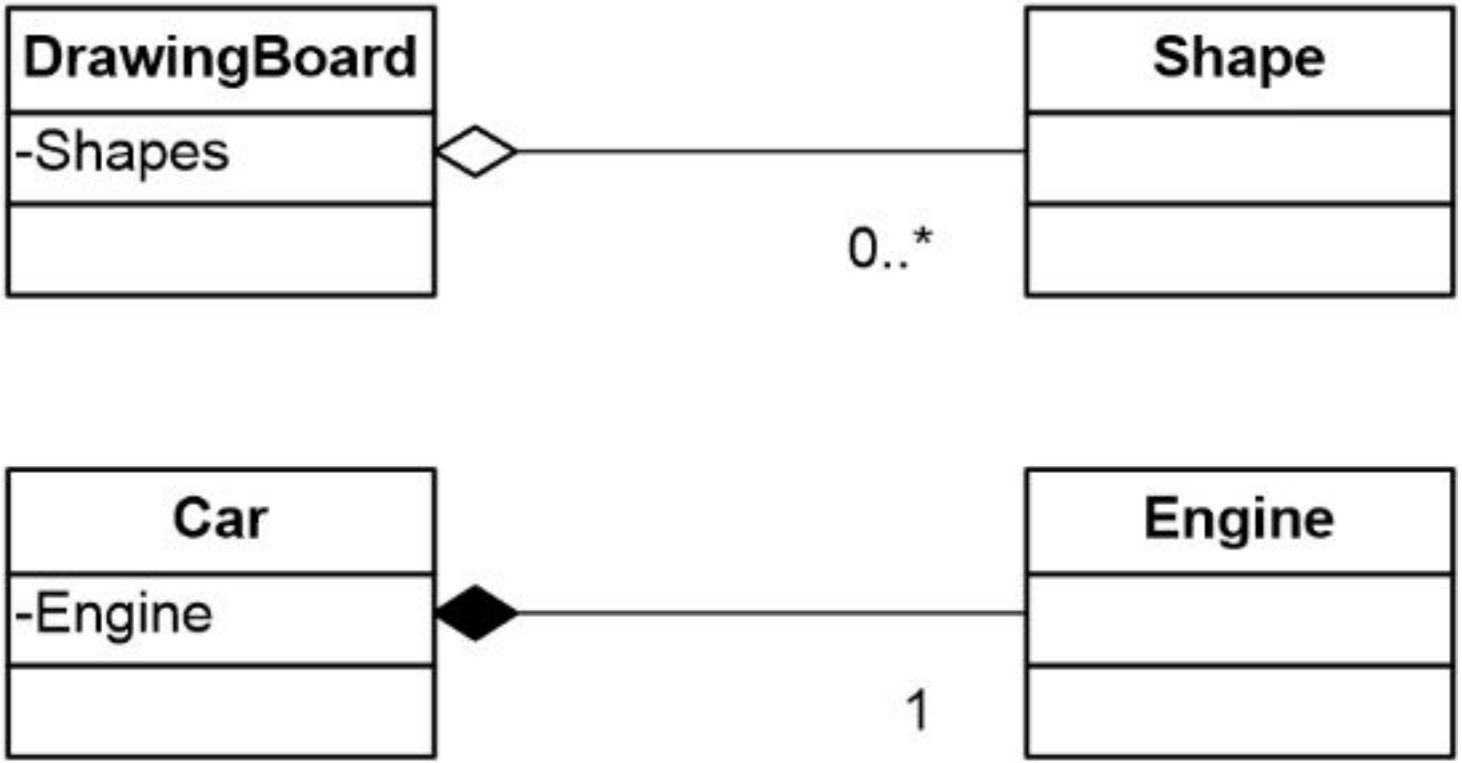
ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Aggregation and Dependency

# Aggregation

- Consider A and B is a Classes
- Aggregation means Object A may contains zero, one or more instance of Object B
- Examples
  - Drawing Board Object may contain zero or more Drawing Shapes Objects
  - Car Object must have 4 Wheels Objects and One Engine Object

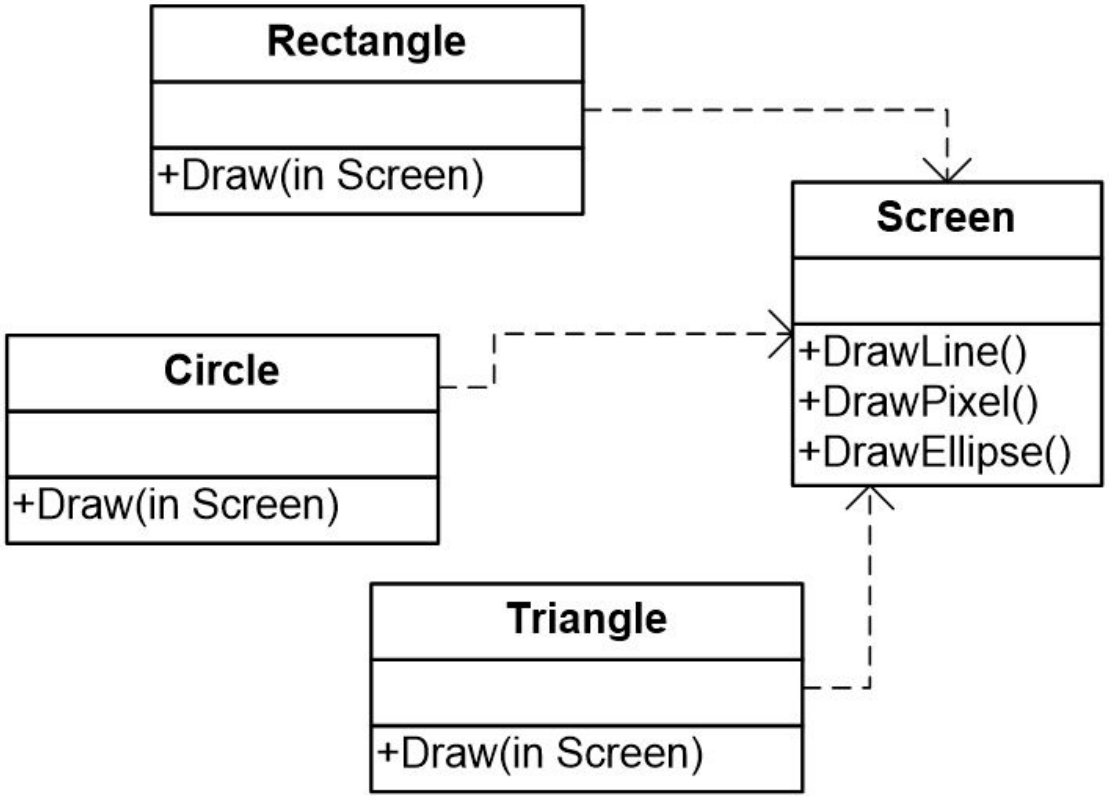
# Aggregation



# Dependency

- Consider A and B is a Classes
- Dependency means Object A may use Object B while performing some operations
- Examples
  - Rectangle Object uses Screen Object while drawing

# Dependency


Rectangle
+Draw(in Screen)
Screen
+DrawLine()
+DrawPixel()
+DrawEllipse()
Circle
+Draw(in Screen)
Triangle
+Draw(in Screen)

# Aggregation Example

```
public class Address {
    private String street, city, state;
    private long zipCode;
    public Address(String street, String city, String state, long zipCode) {
        this.street = street; this.city = city;
        this.state = state; this.zipCode = zipCode;
    }
    public String toString() {
        return street + "\n" + city + ", " + state + " " + zipCode;
    }
}
```

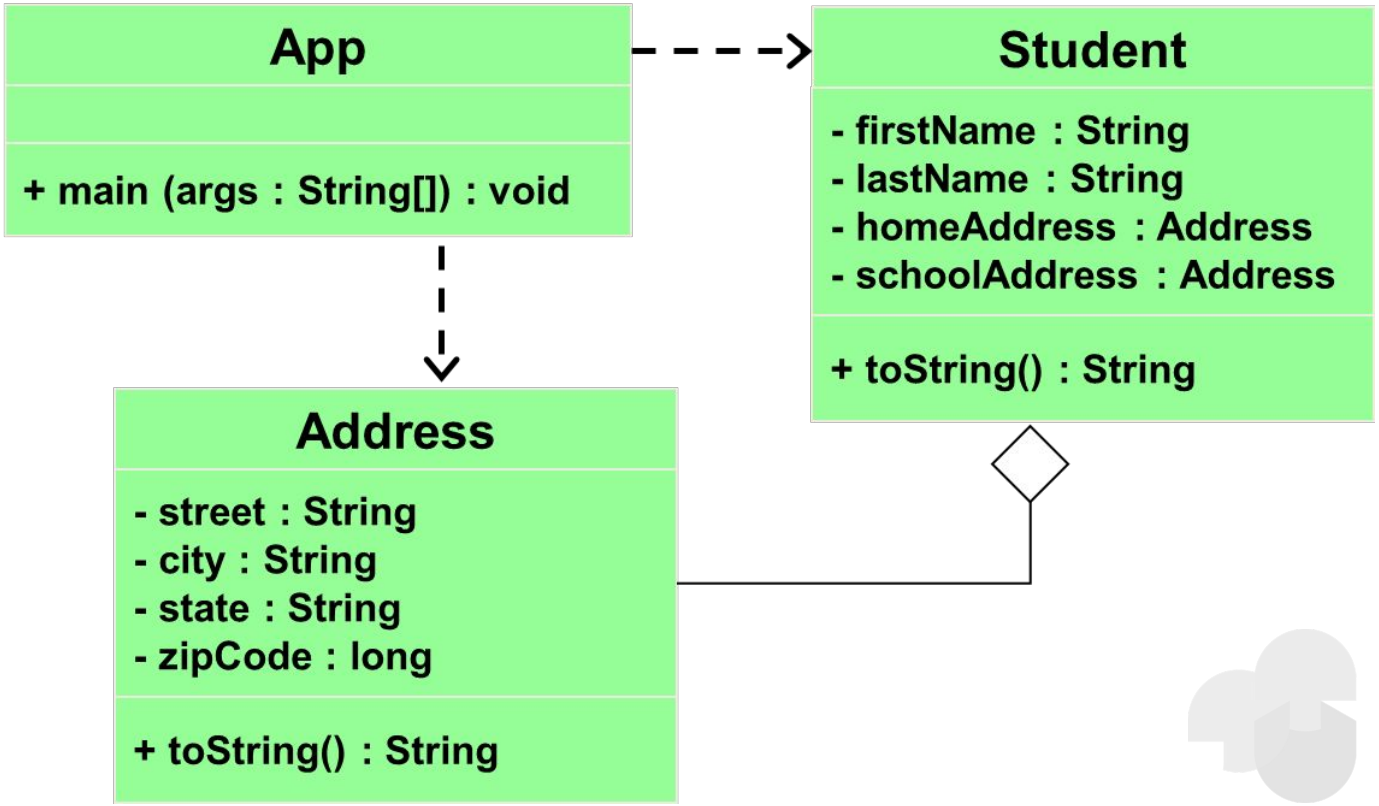
# Aggregation Example

```
public class Student {
    private String firstName, lastName;
    private Address homeAddress, schoolAddress;
    public Student(String firstName, String lastName, Address homeAddress,
                   Address schoolAddress) {
        this.firstName = firstName; this.lastName = lastName;
        this.homeAddress = homeAddress; this.schoolAddress = schoolAddress;
    }
    public String toString() {
        String result;
        result = firstName + " " + lastName + "\n";
        result += "Home Address:\n" + homeAddress + "\n";
        result += "School Address:\n" + schoolAddress;
        return result;
    }
}
```

# Aggregation Example: Using Student Class

```
Address schoolAddress = new Address("800 Lancaster Ave.", "Villanova", "PA",
19085);
Address homeAddress = new Address("21 Jump Street", "Lynchburg", "VA", 24551);
Student s = new Student("John", "Smith", homeAddress, schoolAddress);
System.out.println(s);
```

# Aggregation and Dependency in UML



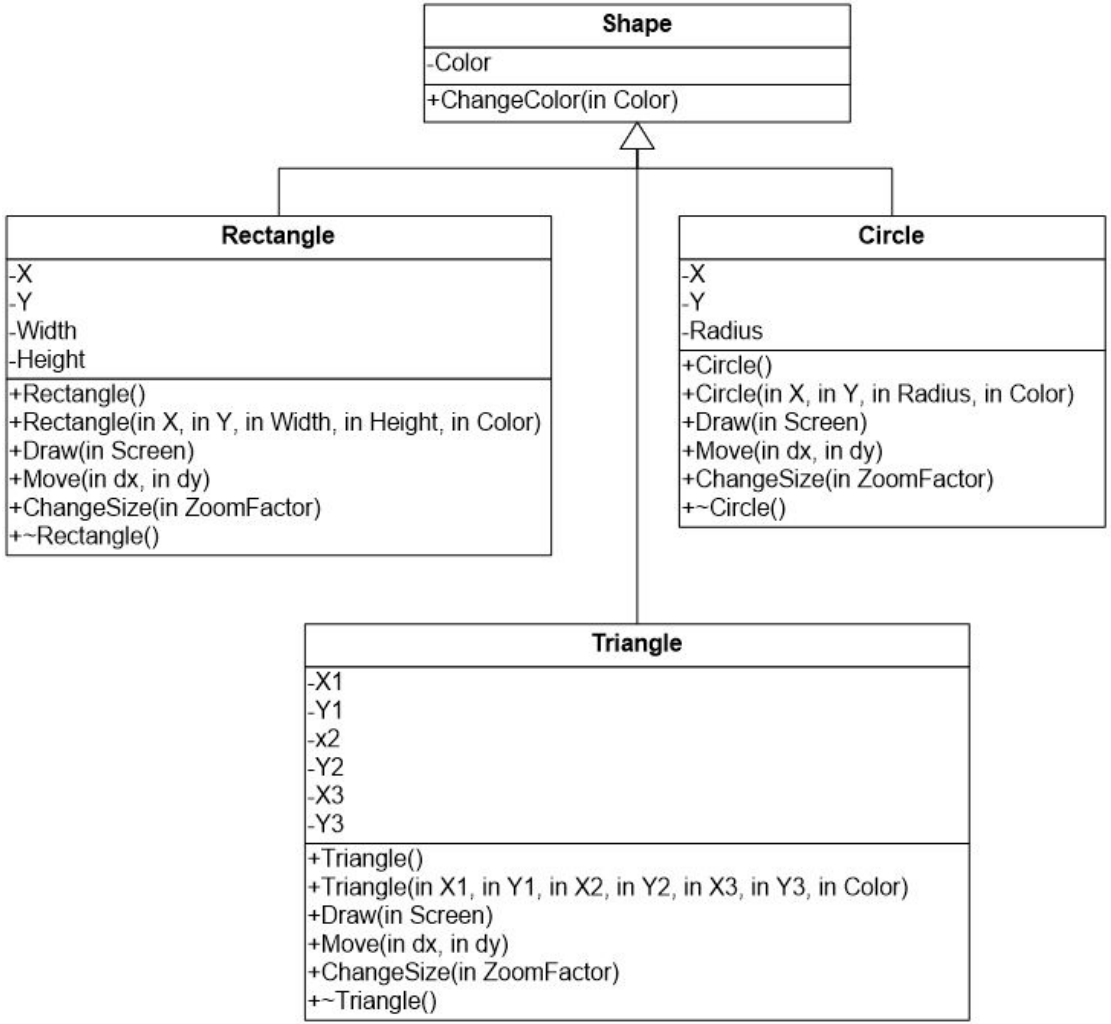
ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Inheritance

# Inheritance

B Class Inherit from A Class means that B Class has All A Class Properties and Methods Besides B Class Properties and Methods

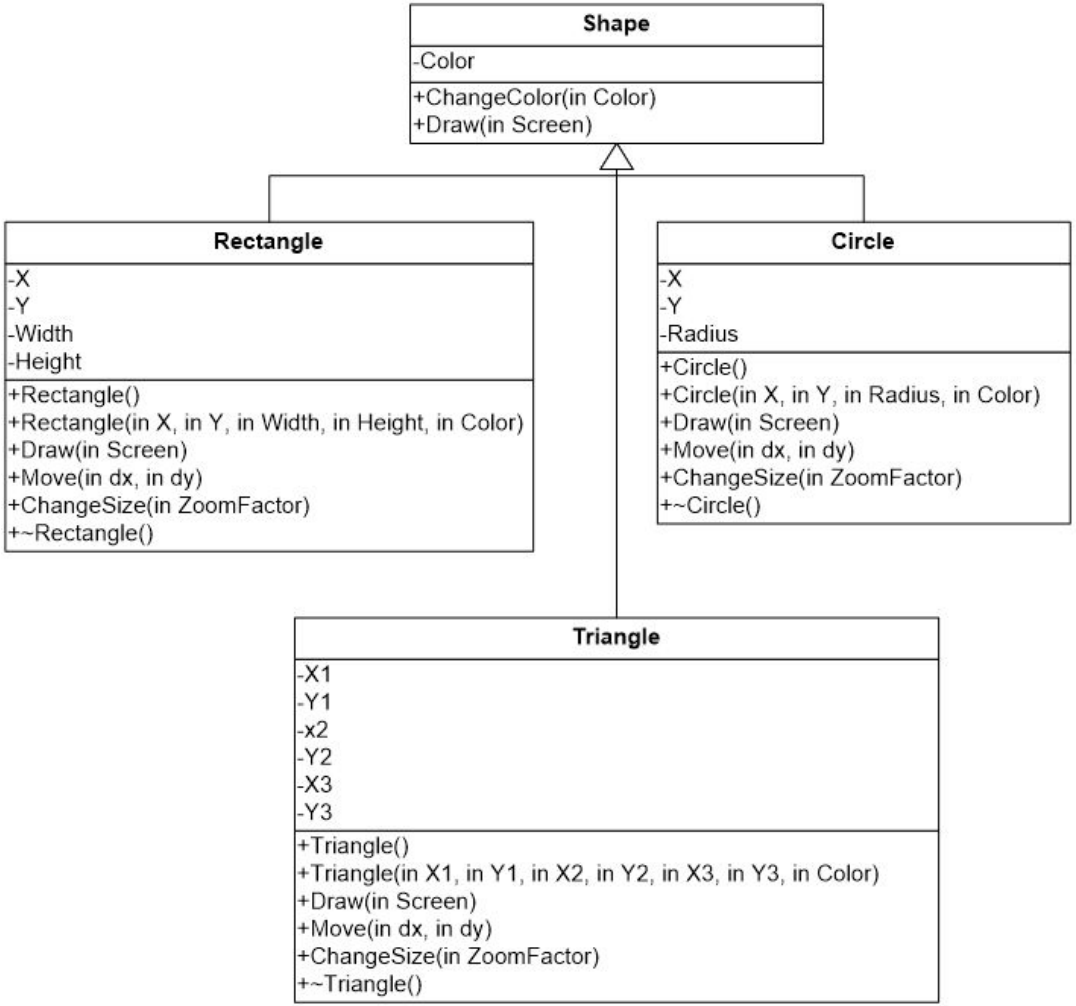
# Inheritance



# Method Overriding

- Draw methods at Rectangle, Circle and Triangle performs different things
- However all of them must configure the same Screen Object with Shape Color before proceeding with drawing

# Method Overriding



# Inheritance Example

```
public class BasicRectangle {
    protected int x1, y1, x2, y2;
    public BasicRectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2;
    }
    public void SetRectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2;
    }
    public String toString() {
        return String.format("[%d,%d,%d,%d]", x1, y1, x2, y2);
    }
}
```

# Inheritance Example

```
public class Rectangle extends BasicRectangle {
    protected String color;
    Rectangle(int x1, int y1, int x2, int y2, String color) {
        super(x1, y1, x2, y2);
        this.color = color;
    }
    public void SetRectangle(int x1, int y1, int x2, int y2, String color) {
        super.SetRectangle(x1, y1, x2, y2);
        this.color = color;
    }
    …
}
```

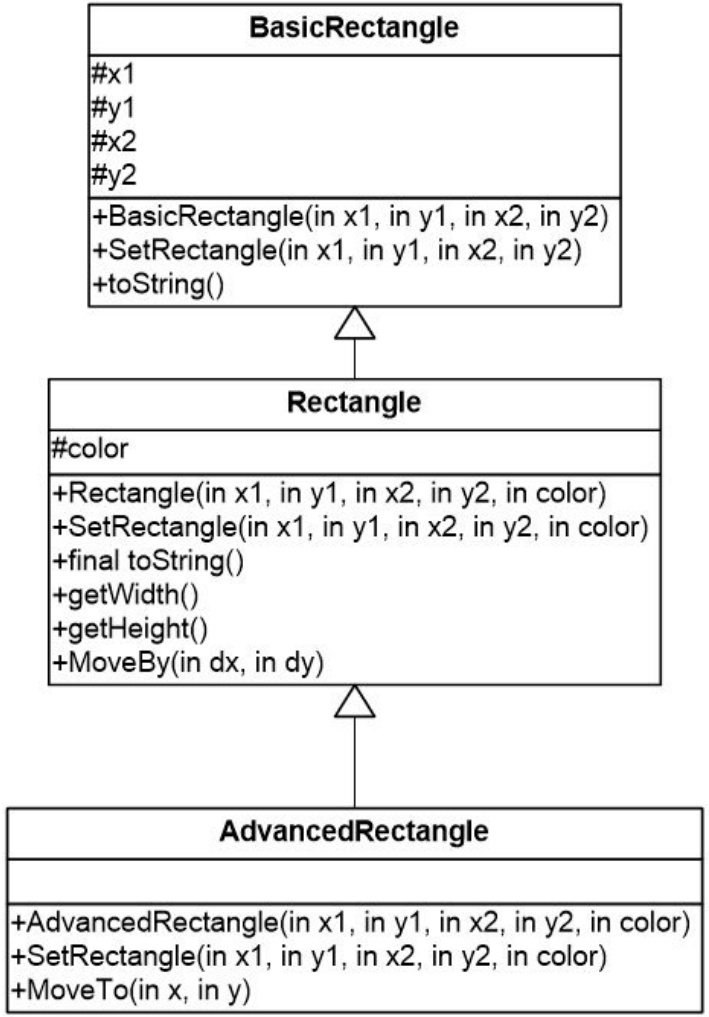
# Inheritance Example

```
public class Rectangle extends BasicRectangle {
    ...
    public int getWidth() {
        return x2 - x1;
    }
    public int getHeight() {
        return y2 - y1;
    }
    public void MoveBy(int dx, int dy) {
        x1 += dx; y1 += dy; x2 += dx; y2 += dy;
    }
}
```

# Inheritance Example

```
public class AdvancedRectangle extends Rectangle {
    public AdvancedRectangle(int x1, int y1, int x2, int y2, String color) {
        super(Math.min(x1, x2), Math.min(y1, y2), Math.max(x1, x2), Math.max(y1, y2), color);
    }
    public void SetRectangle(int x1, int y1, int x2, int y2, String color) {
        super.SetRectangle(Math.min(x1, x2), Math.min(y1, y2), Math.max(x1, x2), Math.max(y1, y2),
color);
    }
    public void MoveTo(int x, int y) {
        x2 = x + getWidth();
        y2 = y + getHeight();
        x1 = x;
        y1 = y;
    }
}
```

# Inheritance in UML



# Inheritance Example

```
BasicRectangle R1 = new BasicRectangle(10, 10, 150, 200);
System.out.println("BasicRectangle : " + R1);
Rectangle R2 = new Rectangle(10, 10, 150, 200, "red");
System.out.println("Rectangle : " + R2);
R2.MoveBy(40, 40);
System.out.println("Rectangle after MoveBy : " + R2);
AdvancedRectangle R3 = new AdvancedRectangle(150, 200, 10, 10, "red");
System.out.println("AdvancedRectangle : " + R3);
R3.MoveBy(40, 40);
System.out.println("AdvancedRectangle after MoveBy : " + R3);
R3.MoveTo(100, 100);
System.out.println("AdvancedRectangle after MoveTo : " + R3);
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Abstract Class

# Abstract Class

- Root class contain some abstract methods
- Abstract methods is tagged with abstract keyword
- Abstract methods is an empty methods

# Abstract Example

```
public abstract class Employee {
    protected String name;
    protected String ID;
    protected double salary;
    public void setName(String name) {this.name = name;}
    public String getName() {return name;}
    public void setID(String ID) {this.ID = ID;}
    public String getID() {return ID;}
    public String toString() { return String.format("(%s - %s)", name, ID);}
    public double getSallary() {return salary;};
    public abstract void setSalary(double salary);
}
```

# Abstract Example

```
public class Developer extends Employee {
    public final double taxes = 0.1;
    @Override
    public void setSalary(double salary) {
        this.salary = salary * (1 - taxes);
    }
}
```

# Abstract Class

```
public class Manager extends Employee {
    public final double taxes = 0.22;
    public final double mobileAllowence = 200;

    @Override
    public void setSalary(double salary) {
        this.salary = (salary + mobileAllowence) * (1 - taxes);
    }
}
```

# Abstract Class

```
Developer d = new Developer();
Manager m = new Manager();
d.setName("john"); d.setID("123");
d.setSalary(1000);
System.out.println(d + " - salary : " + d.getSallary());
m.setName("mark"); m.setID("932");
m.setSalary(5000);
System.out.println(m + " - salary : " + m.getSallary());
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Interface Class

# Interface Class

- A Java interface is a collection of empty methods
- An empty method is a method header without a method body
- A class can implement one or more interfaces

# Interface Example

```
public interface Subject
{
    public void setDegree(double degree);
    public void setMaxDegree(double max);
    public String getGrade();
}
```

# Interface Example

```
public class Course implements Subject{
    private double degree;
    private double max;
    public void setDegree(double degree) {this.degree = degree;}
    public void setMaxDegree(double max) {this.max = max;}
    public String getGrade() {
        double p = 100.0 * degree / max;
        if (p < 50) return "Failed";
        else if (p < 65) return "Pass";
        else if (p < 75) return "Good";
        else if (p < 85) return "Very Good";
        else return "Excellent";
    }
}
```

# Interface Example

```
public class Training implements Subject{
    private double degree;
    private double max;
    public void setDegree(double degree) {this.degree = degree;}
    public void setMaxDegree(double max) {this.max = max;}
    public String getGrade() {
        double p = 100.0 * degree / max;
        if (p < 60) return "Incomplete";
        else if (p < 80) return "Trained";
        else return "Certified";
    }
}
```

# Interface Example

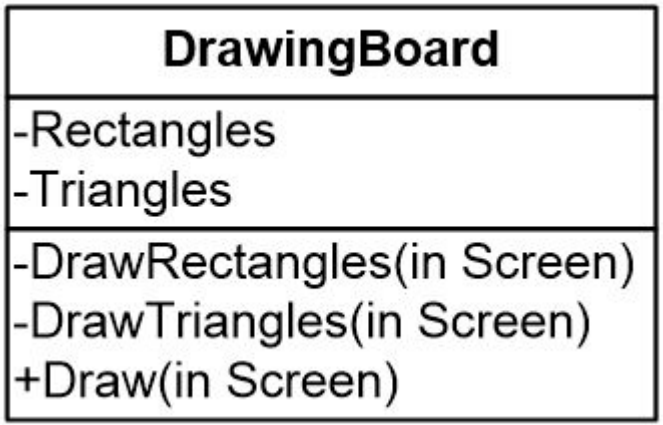
```
Course m = new Course();
m.setDegree(55); m.setMaxDegree(80);
System.out.println(m.getGrade());

Training t = new Training();
t.setDegree(71); t.setMaxDegree(90);
System.out.println(t.getGrade());
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Polymorphism

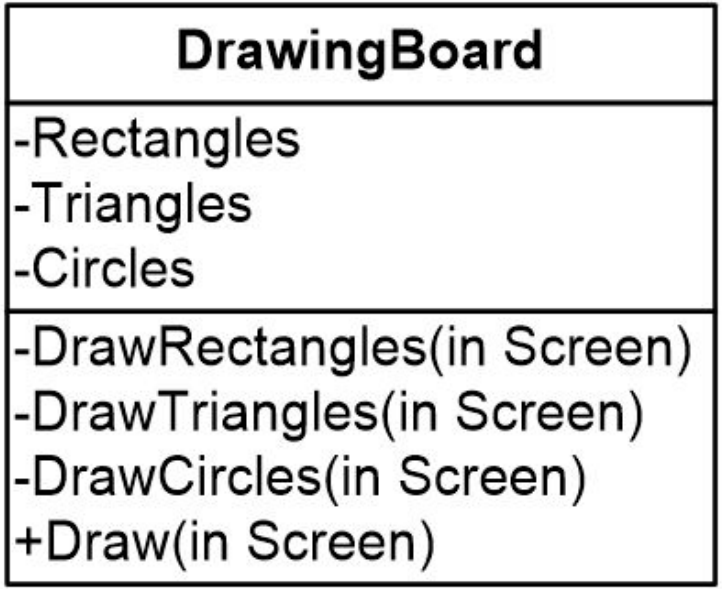
# Polymorphism

- DrawBoard Class allow drawing of Rectangles and Triangles only
- DrawBoard has two lists for each type
- Draw is a Public method that use Private methods DrawRectangles and DrawTriangles



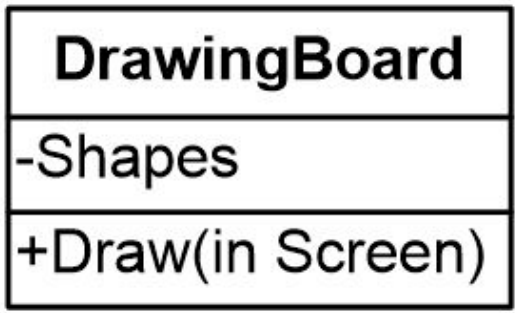
# Polymorphism

- If it is required to support Circles
- Circles List must be added
- DrawCircles method must be added
- Draw method must call DrawCircles

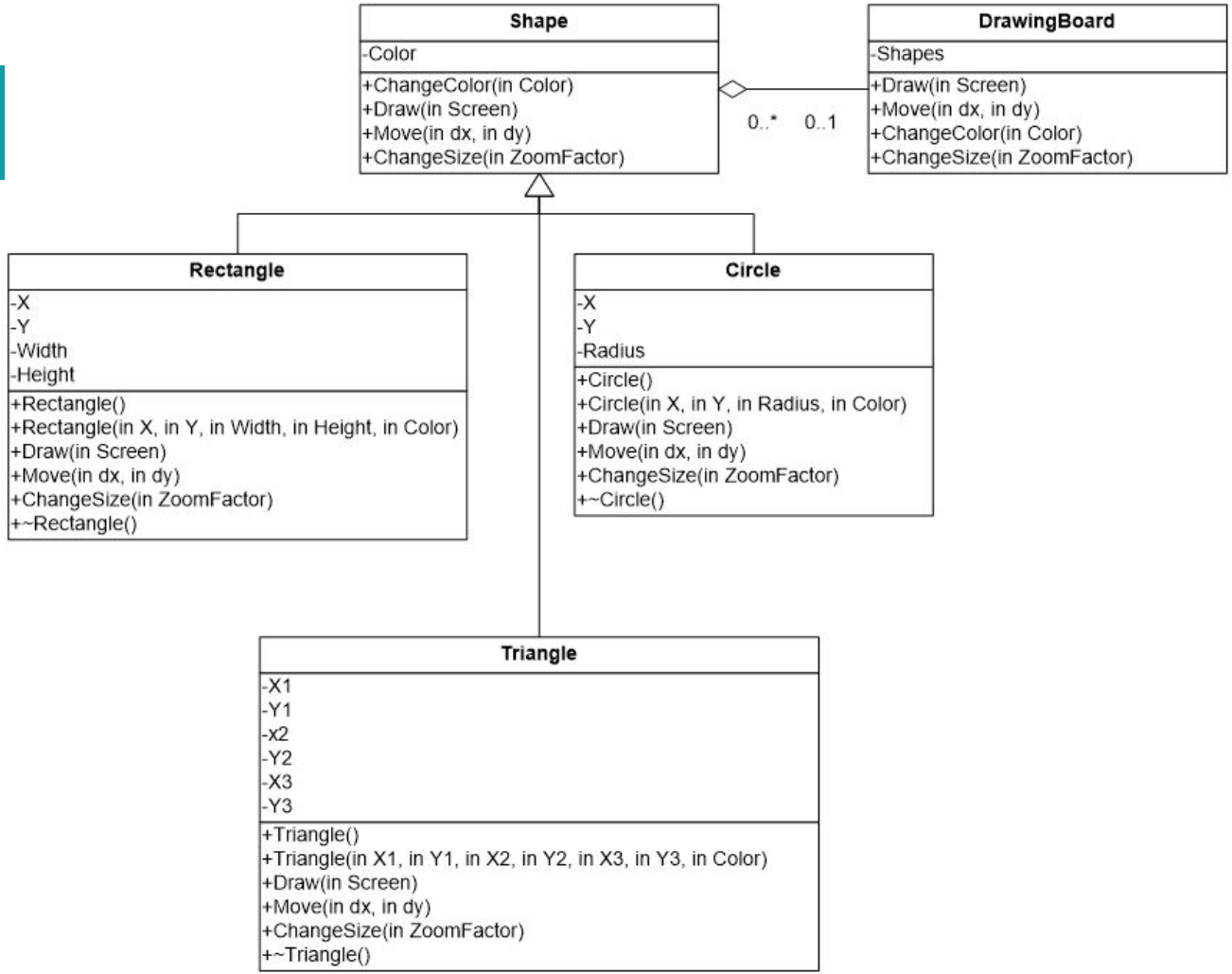


# Polymorphism

- Considering that: (1) All shapes are inherited from Shape Class. (1) Shape Class has a Draw method
- Shapes is a list that store several object considering they inherited from Shape Class
- If other Shape types are added there is no need to modify the DrawingBoard Class



# Polymorphism

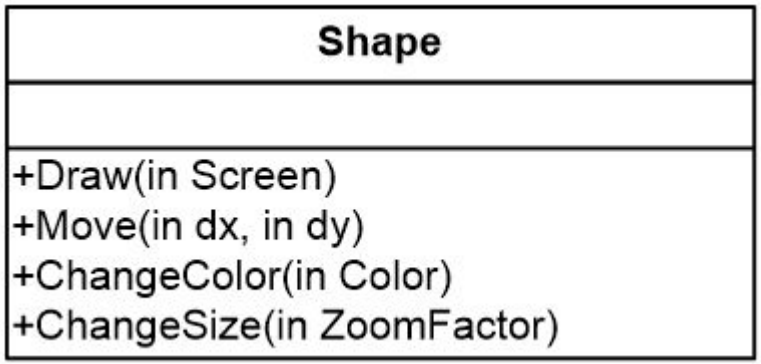


# Abstract Methods and Classes

- Actually Move and ChangSize methods at Shape Class are empty, they are defined to support the Polymorphism
- Empty Methods call abstract methods
- If the class contain some Abstract Methods it called Abstract Class

# Interfaces

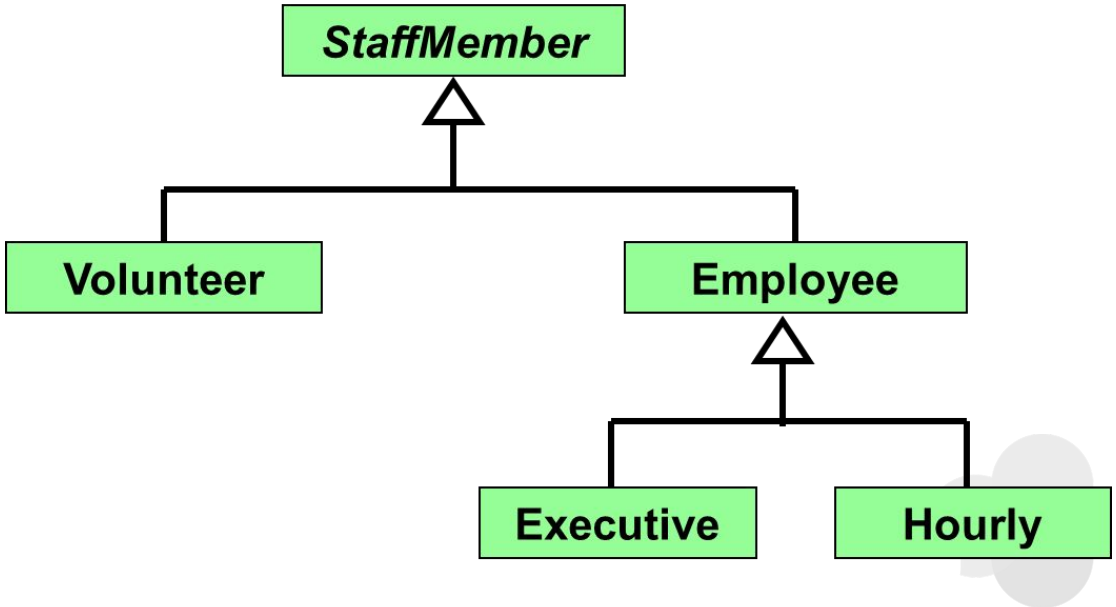
If all root class Methods is empty and the class have no attributes, this class become a Skeleton Class or Interface.



# Polymorphism Example

Staff Member Class can take following forms:

- Employee
  - Executive
  - Hourly
- Volunteer



# Polymorphism Example

```
package staff;
abstract public class StaffMember {
    protected String type;
    protected String name;
    protected String address;
    protected String phone;
    public StaffMember(String name, String address, String phone) {
        this.name = name; this.address = address; this.phone = phone;
    }
    public String getName() {return name;}
    public String toString() {
        String result = "\n--- (" + type +  ") ---";
        result += "\nName: " + name + "\n";
        result += "Address: " + address + "\n";
        result += "Phone: " + phone;
        return result;
    }
    public abstract double pay();
}
```

# Polymorphism Example

```
package staff;
public class Volunteer extends StaffMember {
    public Volunteer(String name, String address, String phone) {
        super(name, address, phone);
        type = "Volunteer";
    }
    public double pay() {
        return 0.0;
    }
}
```

# Polymorphism Example

```
package staff;
public class Employee extends StaffMember {
    protected String socialSecurityNumber;
    protected double rate;
    public Employee(String name, String address, String phone,
          String socialSecurityNumber, double rate) {
        super(name, address, phone);
        this.socialSecurityNumber = socialSecurityNumber;
        this.rate = rate;
        type = "Employee";
    }
    public String toString() {
        return super.toString() + "\nSocial Security Number: " + socialSecurityNumber;
    }
    public double pay() { return rate; }
}
```

# Polymorphism Example

```
package staff;
public class Executive extends Employee {
    private double bonus;
    public Executive (String name, String address, String phone,
              String socialSecurityNumber, double rate) {
        super (name, address, phone, socialSecurityNumber, rate);
        type = "Executive";
        bonus = 0;
    }
    public void awardBonus(double execBonus) { bonus = execBonus; }
    public double pay() {
        double payment = super.pay() + bonus;
        bonus = 0;
        return payment;
    }
}
```

# Polymorphism Example

```
package staff;
public class Hourly extends Employee {
    private int hoursWorked;
    public Hourly(String name, String address, String phone, String
           socialSecurityNumber, double rate) {
        super(name, address, phone, socialSecurityNumber, rate);
        hoursWorked = 0; type = "Hourly";
    }
    public void addHours(int moreHours) { hoursWorked += moreHours; }
    public double pay() {
        double payment = rate * hoursWorked;
        hoursWorked = 0;
        return payment;
    }
    public String toString() {
        return super.toString() + "\nCurrent hours: " + hoursWorked;
    }
}
```

# Polymorphism Example

```
StaffMember[] staffList = new StaffMember[6];
staffList[0] = new Executive("Sam", "123 Main Line", "555-0469", "123-45-6789", 2423.07);
staffList[1] = new Employee("Carla", "456 Off Line", "555-0101", "987-65-4321", 1246.15);
staffList[2] = new Employee("Woody", "789 Off Rocker", "555-0000", "010-20-3040", 1169.23);
staffList[3] = new Hourly("Diane", "678 Fifth Ave.", "555-0690", "958-47-3625", 10.55);
staffList[4] = new Volunteer("Norm", "987 Suds Blvd.", "555-8374");
staffList[5] = new Volunteer("Cliff", "321 Duds Lane", "555-7282");

for (StaffMember staffMember : staffList) {
    System.out.println(staffMember);
}
```

# Polymorphism Example

```
((Executive)staffList[0]).awardBonus (500.00);
((Hourly)staffList[3]).addHours (40);

double total = 0;
for (StaffMember staffMember : staffList) {
    double salary = staffMember.pay();
    String name = staffMember.getName();
    System.out.printf("%s salary is %.2f LE\n", name, salary);
    total += salary;
}
System.out.printf("Total = %.2f LE\n", total);
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Enumeration

# Enumeration

- Java allows definition of enumerated types.
- Enumerated types allow programmer to specify named values instead of using numeric codes.
- Example, you can define enumerated values for the year seasons:

```
enum Season {winter, spring, summer, fall};
```

# Enumeration

```
public class App {
    enum Flavor {vanilla, chocolate, strawberry, coffee}
    public static void main(String[] args) {
        Flavor cone1, cone2;
        cone1 = Flavor.vanilla;
        cone2 = Flavor.chocolate;
        System.out.println("cone1 value: " + cone1);
        System.out.println("cone1 ordinal: " + cone1.ordinal());
        System.out.println("cone1 name: " + cone1.name());
        System.out.println();
        System.out.println("cone2 value: " + cone2);
        System.out.println("cone2 ordinal: " + cone2.ordinal());
        System.out.println("cone2 name: " + cone2.name());
    }
}
```

# Custom Enumeration

```
enum Flavor {
    vanilla(1.5), chocolate(2.0),
    strawberry(2.0), coffee(2.5);
    double price;
    Flavor(double price) {
        this.price = price;
    }
    public double getPrice() {
        return price;
    }
}
```

# Using Custom Enumeration

```
Flavor cone1, cone2;
cone1 = Flavor.vanilla;
cone2 = Flavor.chocolate;
System.out.println("cone1 value: " + cone1);
System.out.println("cone1 ordinal: " + cone1.ordinal());
System.out.println("cone1 name: " + cone1.name());
System.out.println("cone1 price: " + cone1.getPrice());
System.out.println();
System.out.println("cone2 value: " + cone2);
System.out.println("cone2 ordinal: " + cone2.ordinal());
System.out.println("cone2 name: " + cone2.name());
System.out.println("cone2 price: " + cone2.getPrice());
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Iterator

# Iterator

- Used to Navigate through a list of items using
- hasNext and Next and Remove (Optional)
- Implemented by many classes for Example:
  - ArrayList provides an object iterator (E is Object)
  - Scanner implements Iterator<String> (E is String)

# Iterator Interface

```
package java.util.*;

public interface Iterator<E> {

    boolean hasNext();

    E next();

    void remove();
}
```

# Using Iterators

```
import java.util.ArrayList;
import java.util.Iterator;
public class App {
    public static void main(String[] args) {
        ArrayList<String> items = new ArrayList<String>();
        items.add("Hello");
        items.add("Welcome");
        items.add("Goodbye");
        Iterator<String> iterator = items.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

# Using Iterators

```
import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        Scanner scan = new Scanner("824 739 798 743");
        while (scan.hasNext()) {
            System.out.println(scan.next());
        }
    }
}
```

# Custom Iterator

```
import java.util.Iterator;
public class Divisors implements Iterator<Integer> {
    private int D;
    private int N;
    public Divisors(int N) {
        this.N = N; this.D = N;
    }
    public boolean hasNext() {
        return (D != 0);
    }
    public Integer next() {
        for (; D >= 1; D--) {
            if ((N % D) == 0) break;
        }
        int R = D;
        D = D - 1;
        return R;
    }
    public void remove() {}
}
```

```
Divisors D = new Divisors(30);
while (D.hasNext())
    System.out.println(D.next());
```

# Iterable Interface

- Used to Navigate through a list of items using for-each scheme
- Implemented by many classes for Example: ArrayList

```
ArrayList<String> items = new ArrayList<String>();
items.add("Hello");
items.add("Welcome");
items.add("Goodbye");
for (Object S : items) {
    System.out.println(S);
}
```

# Using Iterable Interface

```
import java.util.Iterator;
public class Divisors implements Iterator<Integer>, Iterable<Integer> {
    private int D;
    private int N;
    public Divisors(int N) {
        this.N = N;
        this.D = N;
    }
    public boolean hasNext() {
        return (D != 0);
    }
    public Integer next() {
        for (; D >= 1; D--) {
            if ((N % D) == 0)
                break;
        }
        int R = D;
        D = D - 1;
        return R;
    }
    public void remove() {}
    public Divisors iterator() {
        return this;
    }
}
```

```
Divisors D = new
Divisors(30);
for (Integer I : D)
    System.out.println(I);
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Exception

# Exceptions

- An exception is an object that describes an unusual or erroneous situation
- Exceptions are thrown by a program, and may be caught and handled by another part of the program

# Custom Exception (Mandatory vs Optional)

```
public class OutOfRangeException extends Exception {
    OutOfRangeException(String message) {
        super(message);
    }
}
```

**Mandatory Exception must be handled.**

```
 public class OptionalOutOfRangeException extends RuntimeException {
     OptionalOutOfRangeException(String message) {
         super(message);
     }
 }
```

**Optional Exception.**

# Custom Mandatory Exception

```
import java.util.Scanner;
public class DataEntry {
    public static int getIntInRange(int min, int max) throws OutOfRangeException {
        System.out.print("Enter a value between " + min + " and " + max);
        Scanner scan = new Scanner(System.in);
        int value = scan.nextInt();
        if (value < min || value > max) {
            throw new OutOfRangeException("Input value is out of range.");
        }
        return value;
    }
}
```

```
try {
    int x = DataEntry.getIntInRange(10, 100);
    System.err.println(x);
} catch (OutOfRangeException e) {
    System.out.println(e.getMessage());
}
```

# Custom Optional Exception

```
import java.util.Scanner;
public class DataEntry {
    public static int getIntInRange(int min, int max) throws OptionalOutOfRangeException {
        System.out.print("Enter a value between " + min + " and " + max);
        Scanner scan = new Scanner(System.in);
        int value = scan.nextInt();
        if (value < min || value > max) {
            throw new OptionalOutOfRangeException("Input value is out of range.");
        }
        return value;
    }
}
```

```
int x = DataEntry.getIntInRange(10, 100);
System.err.println(x);
```

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

# Serialization

# Shapes Serialization Example: Shape Class

- Shape
  - Rectangle
  - Circle

```
import java.io.Serializable;

public class Shape implements Serializable{

}
```

# Shapes Serialization Example: Rectangle Class

```
public class Rectangle extends Shape {
    private int x1, y1, x2, y2;
    public Rectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }
    public String toString() {
        return "Rectangle (" + x1 + ", " + y1 + ") - (" + x2 + ", " + y2 + ")";
    }
}
```

# Shapes Serialization Example: Circle Class

```
public class Circle extends Shape {
    private int x, y, radius;
    public Circle(int x, int y, int radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }
    public String toString() {
        return "Circle (" + x + ", " + y + ") - " + radius;
    }
}
```

# Shapes Serialization Example: Write One by One

```
Rectangle r = new Rectangle(10, 10, 100, 100);
Circle c = new Circle(10, 10, 40);
FileOutputStream fos = new FileOutputStream("d:\\serial.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(r);
oos.writeObject(c);
oos.flush();
oos.close();
```

# Shapes Serialization Example: Read One by One

```
FileInputStream fis = new FileInputStream("d:\\serial.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
Rectangle r = (Rectangle) ois.readObject();
Circle c = (Circle) ois.readObject();
ois.close();
System.out.println(r);
System.out.println(c);
```

# Shapes Serialization Example: Write Many

```
ArrayList<Shape> shapes = new ArrayList<Shape>();
shapes.add(new Rectangle(10, 10, 100, 100));
shapes.add(new Circle(10, 10, 40));
FileOutputStream fos = new FileOutputStream("d:\\serial.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(shapes);
oos.flush();
oos.close();
```

# Shapes Serialization Example: Read Many

```
ArrayList<Shape> shapes;
FileInputStream fis = new FileInputStream("d:\\serial.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
shapes = (ArrayList<Shape>) ois.readObject();
ois.close();
for (Shape shape : shapes) {
    System.out.println(shape);
}
```