Software Development and Best Practices

Part 1 : Object-oriented programming Revision

Mahmoud Khalaf Saeed

Section (1)

THIS KEYWORD

```
package SeconedSection;
public class LearnThisKeyword
{
    int i = 5 ;
    static double k = 10 ;
    public int SetGetI (int i)
    {   //i = i ;     // where to from ?
        this.i = i;
        return  i ;
    }
    public double SetGetK ( double k ) {
        this.k  =  k  ;
        return k  ;
    }

    public static void main(String[] args)
    {
        LearnThisKeyword tes  = new LearnThisKeyword() ;
        System.out.println("the value of i is " + tes.SetGetI(5)) ;
        System.out.println("the value of k is " + tes.SetGetK(5.5)) ;
        System.out.println("static variable can accesed as "
                + "className.varname " + LearnThisKeyword.k) ;
    }
   }
```

The ***this*** keyword is used to refer to a calling object itself.

A hidden static variable can be accessed simply by using the ClassName.Static Variable

```java
package SeconedSection;
public class LearnThisKeyword
{
    int value ;
    public LearnThisKeyword() {
        System.out.println("Hello from constructor!");
    }
    public LearnThisKeyword( int value) {
        this();
        this.value = value;
    }
    public void n (){
       System.out.println("Greeting Message From N");
    }
    public void m (){
        System.out.println("Greeting Message From M");
        this.n();
    }
    public static void main(String[] args) {
        LearnThisKeyword tes  = new LearnThisKeyword(10);
        tes.m();
    }
}
```

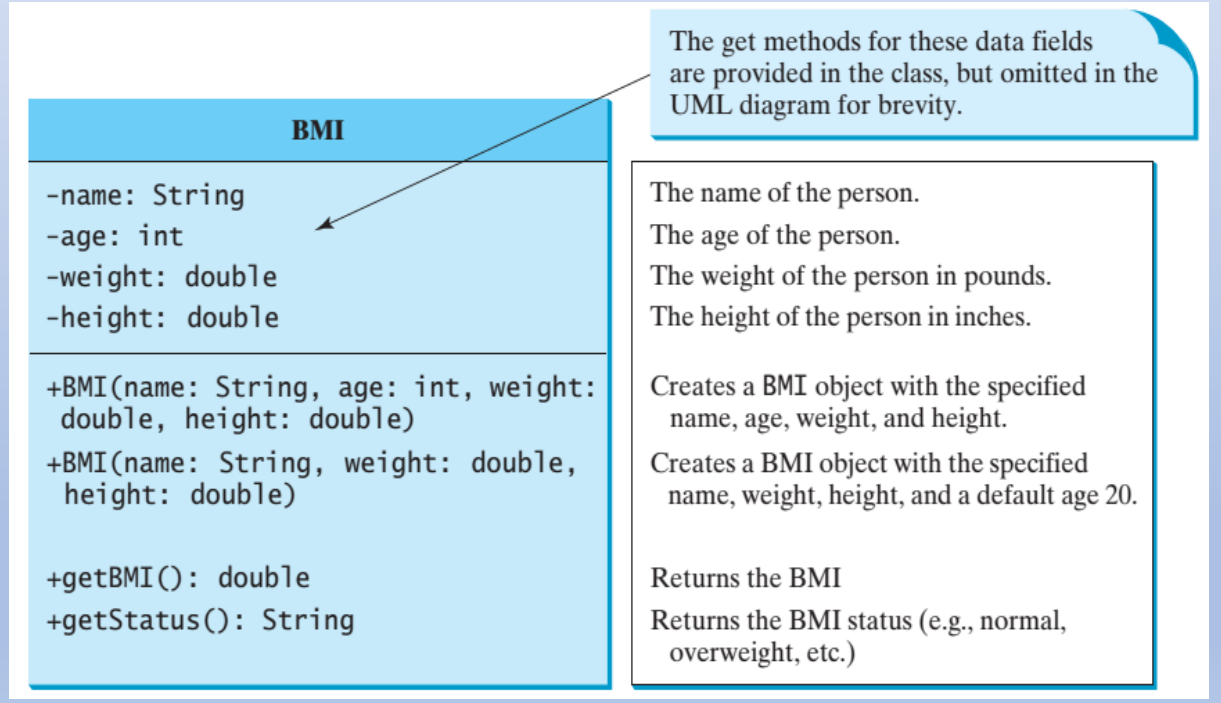
Another common use of the this keyword is to enable a constructor to invoke another constructor of the same class.

Another common use of the this keyword is to enable a function to invoke another function

COMPUTING BODY MASS INDEX

Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters.

BMI	Interpretation
below 16	seriously underweight
16–18	underweight
18–24	normal weight
24–29	overweight
29–35	seriously overweight
above 35	gravely overweight



Inheritance

INHERITANCE

Object-oriented programming allows you to derive new classes from existing classes. This is called ***inheritance***.

Inheritance is an important and powerful feature in Java for **reusing software**.

Classes (circles, rectangles, and triangles) may have many common features

its better to design these classes as a more generalized class that can be extended later in more specialized classes. .

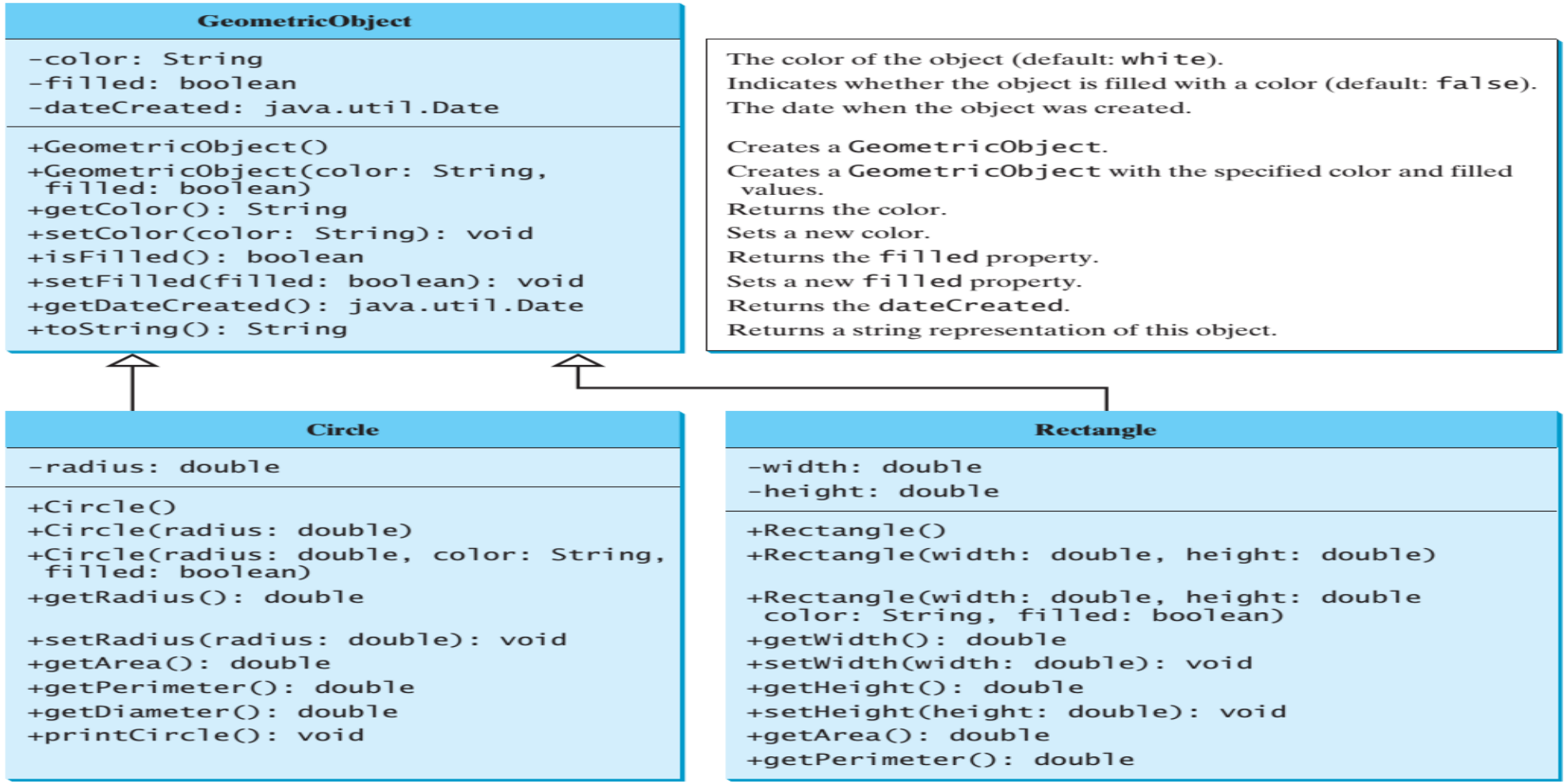
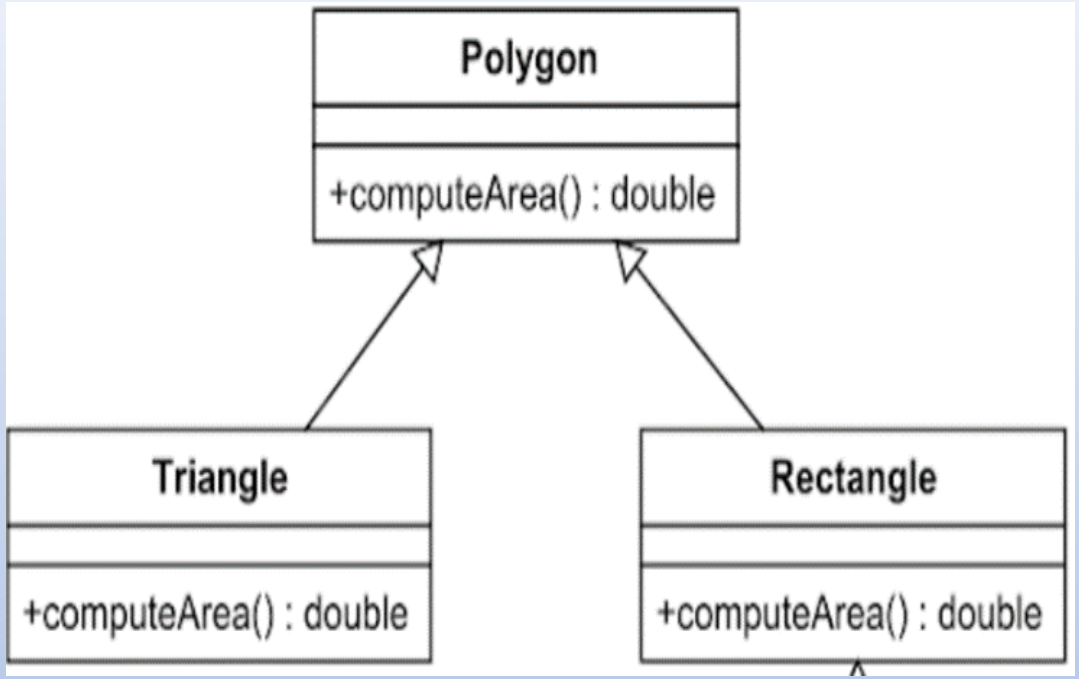


FIGURE 11.1 The `GeometricObject` class is the superclass for `Circle` and `Rectangle`.

INHERITANCE



Circle is the child , extended , or derived class **GeometricObject** is the parent , base or super class

The keyword **extends** tells the compiler that the **Circle** class extends the **Geometric- Object** class, thus inheriting the methods **getColor**, **setColor**, **isFilled**, **setFilled**, and **toString**.

```java
package MyPackage;
public class GeometricObjects
{
 private String color = "white";  //Encapsulation
 private boolean filled;
 private java.util.Date dateCreated;

/** Construct a default geometric object */
 public GeometricObjects()
                {
                    dateCreated = new java.util.Date();
                }

 public GeometricObjects(String Color, boolean filled)
          {
              this.color = color;
              this.filled = filled;
          }

 /** Return color */
 public String getColor()
          {
              return color;
          }
 /** Set a new color */
 public void setColor(String color)
          {
              this.color = color;
          }
```

```
        {
            this.color = color;
        }

public boolean isFilled()
        {
            return filled;
        }

/** Set a new filled */
public void setFilled(boolean filled)
        {
            this.filled = filled;
        }

/** Get dateCreated */
public java.util.Date getDateCreated()
        {
            return dateCreated;
        }

/** Return a string representation of this object */
public String toString()
        {
            return "created on " + dateCreated + "\ncolor: " + color +
            " and filled: " + filled;
            }
}
```

```java
package MyPackage;
public class Circle11 extends GeometricObjects
    {
       private double radius;

public Circle11(double radius)
    {
       this.radius = radius;
    }

public Circle11(double radius, String color, boolean filled)
    {
        this.radius = radius;
        setColor(color);    //inherited as a private ==> accessed through set
        setFilled(filled);
    }

 /** Return radius */
 public double getRadius()
     {
       return radius;
     }

 /** Set a new radius */
 public void setRadius(double radius)
     {
       this.radius = radius;
     }
```

```java
/** Return area */
public double getArea()
    {
      return radius * radius * Math.PI;
    }
/** Return diameter */
public double getDiameter()
    {
      return  2 * radius;
    }

/** Return perimeter */
public double getPerimeter()
    {
      return  2 * radius * Math.PI;
    }

/* Print the circle info */
public void printCircle()
    {
      System.out.println("The circle is created " + getDateCreated() +
                         " and the radius is " + radius);
    }

}
```

```java
package MyPackage;
public class Rectangle11 extends GeometricObjects
{
    private double width;
    private double height;

public Rectangle11(double width, double height)
    {
        this.width = width;
        this.height = height;
    }

 public Rectangle11(double width, double height, String color,boolean filled)
    {
        this.width = width;
        this.height = height;
        setColor(color);
        setFilled(filled);
    }

 /** Return width */
 public double getWidth()
    {
        return width;
    }

 /** Set a new width */
 public void setWidth(double width)
```

```
public void setWidth(double width)
    {
       this.width = width;
    }

/** Return height */
public double getHeight()
    {
       return height;
    }

/** Set a new height */
public void setHeight(double height)
    {
       this.height = height;
    }

/** Return area */
public double getArea()
    {
       return width * height;
    }

/** Return perimeter */
public double getPerimeter()
    {
       return 2 * (width + height);
    }
}
```

```
package MyPackage;
public class TestCircleRectangle
{
    public static void main(String[] args)
    {
       Circle11 circle = new Circle11(1);

    System.out.println("A circle " + circle.toString() );
    System.out.println("The radius is " + circle.getRadius());
    System.out.println("The area is " + circle.getArea());
    System.out.println("The diameter is " + circle.getDiameter() );

    Rectangle11 rectangle = new Rectangle11( 2 ,4 );

    System.out.println("\nA rectangle " + rectangle.toString());
    System.out.println("The area is " + rectangle.getArea());
    System.out.println("The perimeter is " + rectangle.getPerimeter());
    }
}
```

EXCEPTED OUTPUT

```
run:
A circle created on Sun Oct 17 09:54:26 EET 2021
color: white and filled: false
The radius is 1.0
The area is 3.141592653589793
The diameter is 2.0

A rectangle created on Sun Oct 17 09:54:26 EET 2021
color: white and filled: false
The area is 8.0
The perimeter is 12.0
```

POINTS REGARDING TO INHERITANCE

- Not all *is-a* relationships should be modeled using inheritance.
- For example, a square is a rectangle, but you should not define a **Square** class to extend a **Rectangle** class,
- because there is nothing to extend (or supplement) from a rectangle to a square.
- Rather you should define a **Square** class to extend the **GeometricObject** class.
- For class **A** to extend class **B**, **A** should contain more detailed information than **B**.

Inheritance is used to model the *is-a* relationship. Do not blindly extend a class just for the sake of reusing methods.

For example, it makes no sense for a **Tree** class to extend a **Person** class, even though they share common properties such as height and weight.

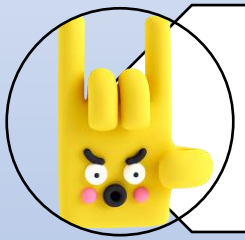
A subclass and its superclass must have the *is-a* relationship.

Some programming languages allow you to derive a subclass from several classes. This capability is known as ***multiple inheritance***.

Java, however, does not allow multiple inheritance. A Java class may inherit directly from only one superclass. This restriction is known as ***single inheritance***.

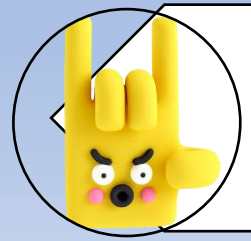
THE SUPER KEYWORD

It is a reference variable which is used to refer immediate parent class object. Whenever you **create the instance** of subclass, an instance of parent class **is created implicitly** which is referred by super reference variable



super can be used to refer parent class instance variable.

super can be used to invoke parent class method.



super() can be used to invoke parent class constructor.

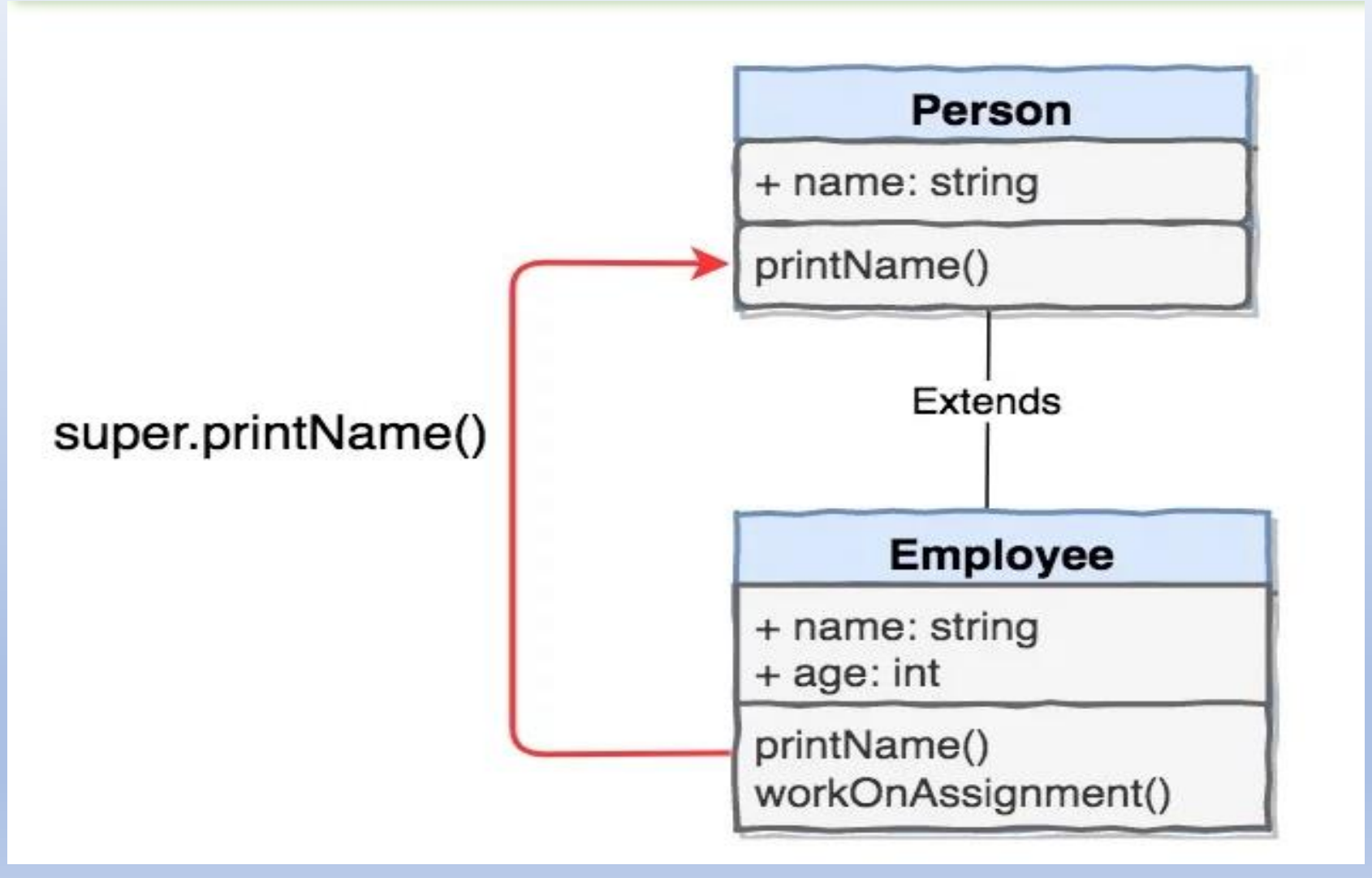
```java
package inheritance;
//mammals =>  [elephants , rabbits , whales]

public class Mammals
{
    public String color;
    public int speed ;
    public int weight ;

    public Mammals(String color, int speed, int weight)
    {
        this.color = color;
        this.speed = speed;
        this.weight = weight;
    }

    public void printweight ()
    {
        System.out.println("the weight of mammal is " + weight);
    }
}
```

```java
package inheritance;
public class Elephants extends Mammals
{

    public Elephants()
    {   // invoking parent constructor
        super("black"  , 100 , 200000);
        System.out.println("elephant is created");
    }

    public  void eat ()
    {
        System.out.println(" elephans eats fruit and tree bark");
        System.out.println("the color is " + super.color); // parent instance var
        super.printweight();  // invoking parent method
    }
}
```

```java
package inheritance;
public class TestMammals
{
    public static void main(String[] args)
    {
        Elephants elephant = new Elephants();
        elephant.eat();
    }
}
```

```
run:
elephant is created
elephans eats fruit and tree bark
the color is black
the weight of mammal is 200000
```

CODE IT


Person
+ name: string
printName()
Extends
super.printName()
Employee
+ name: string
+ age: int
printName()
workOnAssignment()

AGGREGATION

- If a class have an entity reference, it is known as **Aggregation**.
- **Aggregation** represents HAS-A relationship.
- Employee object contains many information such as id, name, email
- It contains one more object named address, which contains its own information such as city, state, country, zipcode etc.
- In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

```java
package AGGREGATION;
public class Employee
{
    int id;
    String name;
    Address address; //entity reference represented as a class

        public Employee(int id, String name,Address address)
        {
            this.id = id;
            this.name = name;
            this.address=address;
        }

        void display()
        {
            System.out.println("Emp ID is : "+id+" and Emp NAME is : "+name);
            System.out.println("Emp ADDRESS is : " +address.city+" "+address.state+" "
                                                    +address.country);
        }
}
```

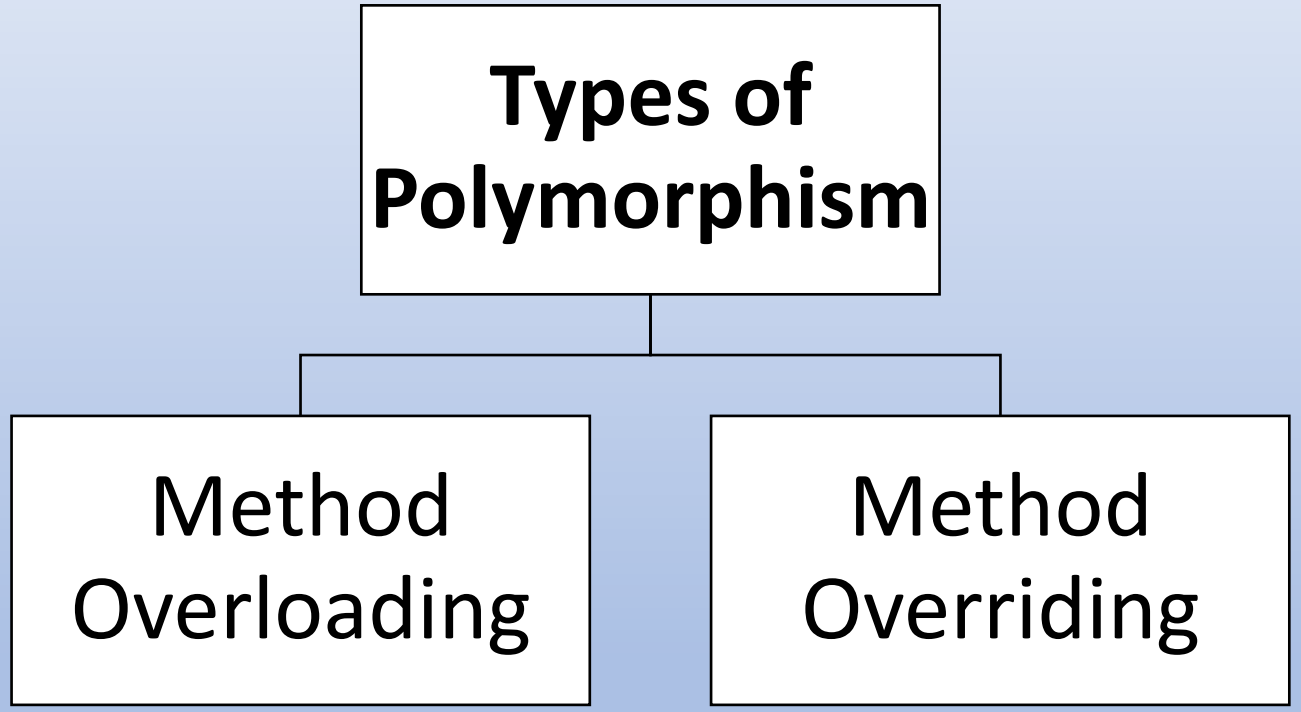
```java
package AGGREGATION;
public class Address
{
    String city ;
    String state ;
    String country ;

    public Address(String city, String state, String country)
    {
        this.city = city;
        this.state = state;
        this.country = country;
    }
}
```

```java
package AGGREGATION;
public class TestAggregation
{
    public static void main(String[] args)
    {
        //creating two objects of class Address
        Address address1=new Address("Mallawi","Minya","Egypt");
        Address address2=new Address("Samalout","Minya","Egypt");

        //creating two objects of class Employee
        Employee e1=new Employee(101010,"ALi",address1);
        Employee e2=new Employee(202020,"Omar",address2);

        e1.display();
        e2.display();
    }
}
```

```
run:
Emp ID is : 101010 and Emp NAME is : ALi
Emp ADDRESS is : Mallawi Minya Egypt
Emp ID is : 202020 and Emp NAME is : Omar
Emp ADDRESS is : Samalout Minya Egypt
```

POLYMORPHISM

Polymorphism in Java is the ability of an object to take many forms.



METHOD OVERLOADING IN JAVA

- If a class has multiple methods having **same name** but different in parameters, it is known as **Method Overloading**.

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

In Java, Method Overloading is not possible by changing the return type of the method only [function Re-defintion].

```java
package Polymorphism;
public class Box
{
    public  int length , width , height ;
    public Box(int length, int width, int height)
    {
        this.length = length; this.width = width; this.height = height;
    }
    public int  getVolume ()
    {
        return length  * width  * height ;
    }
    /*public double getVolume ()
     {     // method re-definition
         return length  * width  * height ;
     }*/
    public int  getVolume (int scale )
    {
        return length  * width  * height * scale ;
    }
    public double getVolume (double scale )
    {
        return length  * width  * height * scale ;
    }
}
```

```java
package Polymorphism;
public class TestBox
    {
        public static void main(String[] args)
        {
            Box b = new Box(5, 3, 2);
            System.out.println("Volume is " + b.getVolume());
            System.out.println("scaled Volume is " + b.getVolume(10));
            System.out.println("scaled double Volume is " + b.getVolume(1.5));
        }
    }
```

```
run:
Volume is 30
scaled Volume is 300
scaled double Volume is 45.0
```

METHOD OVERRIDING IN JAVA

Overriding occurs If subclass (child class) has the **same method** as declared in the parent class In other words, If a subclass provides the **specific implementation** of the method

There are two ways to overload the method in java

- ❑ Method must have the same parameter as in the parent class
- ❑ There must be an IS-A relationship (inheritance).
- ❑ Method must have the same name as in the parent class

Java method overriding is mostly used in Runtime Polymorphism

```java
package Polymorphism;
    public class TestBank
    {
        public static void main(String[] args)
        {
            CIB c = new CIB();
            Ahly a = new Ahly();
            Misr m = new Misr();

            System.out.println("value added tax in CIB  bank is : " +c.valueAddedTax(1000));
            System.out.println("value added tax in Ahly bank is : " +a.valueAddedTax(1000));
            System.out.println("value added tax in Misr bank is : " +m.valueAddedTax(1000));
        }
    }
```

```
run:
value added tax in CIB  bank is : 80.0
value added tax in Ahly bank is : 70.0
value added tax in Misr bank is : 60.0
```

```java
package Polymorphism;
public class CIB extends Bank
    {
    @Override   // optional
      public double valueAddedTax(double amount)
       {
          return 0.80 * amount ;
       }
    }
```

```java
package Polymorphism;
    public class Misr extends Bank
    {
    public double valueAddedTax(double amount)
        {
           return amount * 0.60 ;
        }
    }
```

```java
package Polymorphism;
    public class Ahly extends Bank
    {
       public double valueAddedTax(double amount)
       {
          return 0.70 *amount;
       }
    }
```

```java
package Polymorphism;
public class Bank
    {
     public double valueAddedTax ( double amount)
     {
         return amount;
     }
    }
```

JAVA FINAL KEYWORD

- Java final keyword is a non-access specifier that is used to restrict a class, variable, and method to be changed .

Stop value change [variable]
Stop method overriding [method]
Stop class inheritance [class]

```
package Polymorphism;
public class finalKeyword
{
    final int x = 20 ;

    void changeX ()
    {
        x = x + 20  ;
       //cannot assign a value to final variable x
    }
}
```

```java
package Polymorphism;
public class finalKeyword
{
    final void printMsg()
    {
        System.out.println("content of message can't be changed ");
    }
}
```

```java
package Polymorphism;
public class Xchange extends finalKeyword
{
    void printMsg()
    {
        /*printMsg() in Xchange cannot override printMsg()
                in finalKeyword overridden method is final */
    }
}
```

```java
package Polymorphism;
final public class finalKeyword
{
    final void printMsg()
    {
        System.out.println("content of message can't be changed ");
    }
}
```

```java
package Polymorphism;
public class Xchange extends finalKeyword
{
    //can't inherit from final finalKeyword
}
```

ASSIGNMENT

Design a class named **Person** and its two subclasses named **Student** and **Employee**.

Make **Faculty** and **Staff** subclasses of **Employee**.

A person has a name, address, phone number, and email address.

A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant.

An employee has an office, salary and date hired.

Define a class named **MyDate** that contains the fields **year**, **month**, and **day**. (hint aggregation here)

A faculty member has office hours and a rank.

A staff member has a title.

Override the toString method in each class to display the class name and the person's name.

Draw the UML diagram for the classes.

Implement the classes.

Write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their toString() methods.

ASSIGNMENT

Create a class called student which contain the following data fields:

- Student name , Private studentID , studentGPA
- entity reference **subject** [subjectID , subjectName , subjectHours]
- entity reference **address** [streetNO , city , country]

Add the appropriate methods that you can use for dealing with this class :

Have a good day