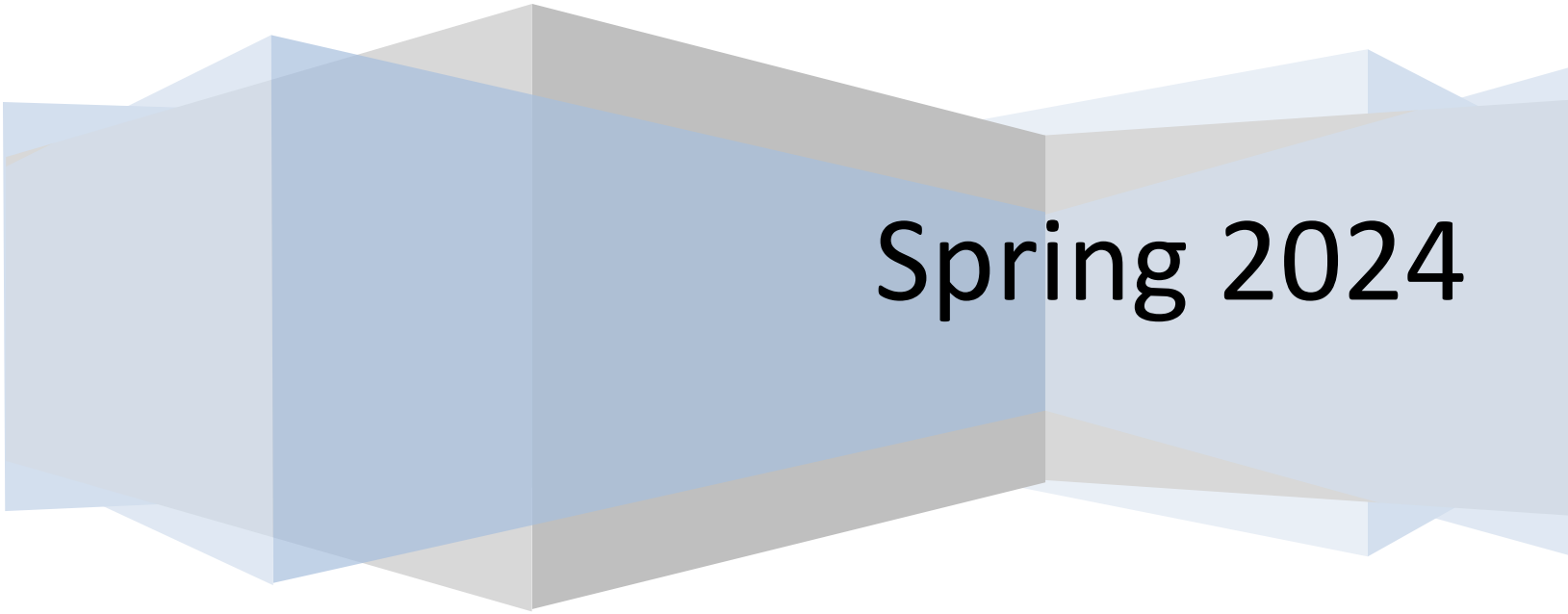
Northern University, Nowshera

Polymorphism

Week # 13 - Lecture 25 - 26



Spring 2024

Learning Objectives:

- **Assignment Solution (Week 12)**
- **Polymorphism**
- **Compile time Polymorphism**
- **Method overloading**
- **Operator overloading**
- **Run time/Dynamic Polymorphism**
- **Method overriding**
- **Overloading VS overriding**
- **Static polymorphism VS Dynamic Polymorphism**
- **Home task for practice**

Assignment Solution

Q: Write code for the following classes.

You are required to implement a system where information of authors with their books has been stored using the concept discussed in this lesson (Association, aggregation and composition).

Author class contains author name (String), email (String), total number of written books with following book detail.

Book class has book ID (String), Book Title (String), price (float) and publisher (String) has attributes.

Associate these two classes with each other (according to your understanding) in a way that user can get the information of author with detail of his total written books, like book title, publisher, id and price. Demonstrate your program in main() function by creating an object of class author with at least 3 books.

Note: Write setter and getter functions for both classes and also write appropriate constructors.

Solution:

```
class Book{
    String title, publisher, ID;
    float price;
    void setBook(){
        System.out.println("Enter Book title: ");
        Scanner s=new Scanner(System.in);
        title=s.nextLine();
        Scanner s1=new Scanner(System.in);
        System.out.println("Enter Book Publisher: ");
        publisher=s1.nextLine();

        System.out.println("Enter Book Price: ");
```

```
        price=s.nextFloat();

        System.out.println("Enter Book ID: ");
        ID=s1.nextLine();
    }
    void getBook()
    {
        System.out.println(title+"\t"+publisher+"\t"+ID+"\t"+price+"RS\n");
    }
}

class Author {
    String name, email;
    int totalBooks;
    Book books[];


    void setAuthor(){
        System.out.println("Enter Author name: ");
        Scanner s=new Scanner(System.in);
        name=s.nextLine();
        System.out.println("Enter Author Email: ");
        email=s.nextLine();
        System.out.println("Enter total books written: ");
        totalBooks=s.nextInt();
        books=new Book[totalBooks];
        for (int i=0;i<books.length; i++) {
            books[i]=new Book();
            books[i].setBook();
        }
    }
    void getAuthor()
    {
        System.out.print(name+"\t"+email+"\t"+totalBooks+"\t");

        for (Book b: books) {
            b.getBook();
        }
    }


}

class MAINCLASS{
    public static void main(String[] args) {
        Author a=new Author();
        a.setAuthor();
        a.getAuthor);
    }
}
```

Output

```
Enter Author name:
Robert Lafore
Enter Author Email:
robert@gmail.com
Enter total books written:
2
Enter Book title:
Object oriented programming
Enter Book Publisher:
abc publisher
Enter Book Price:
400
Enter Book ID:
121BIIIT
Enter Book title:
Programming Fundamentals
Enter Book Publisher:
xyz publishser
Enter Book Price:
500
Enter Book ID:
UIIT234
Robert Lafore    robert@gmail.com 2

Object oriented programming      abc publisher    121BIIIT 400.0 RS

Programming Fundamentals xyz publishser   UIIT234  500.0 RS
```

Polymorphism click here for video

Polymorphism is an important concept of object-oriented programming. It simply means more than one forms, that is, the same entity (method or operator or object) behaves differently in different scenarios. For example: The + operator in Java is used to perform two specific functions. When it is used with numbers (integers and floating-point numbers), it performs addition.

```
int a = 5;
int b = 6;
int sum = a + b;      //  Output = 11
```

And when we use + operator with strings, it performs string concatenation. For example,

```
String firstName = "abc ";
String lastName = "xyz";
name = firstName + lastName;      //  Output = abc xyz
```

Real life example of polymorphism: A person at the same time can have different characteristic, like a man at the same time is a father, a husband, an employee, so the same person posses different behavior in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming.

- Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.
- The word “poly” means many and “morphs” means forms, so it means many forms.

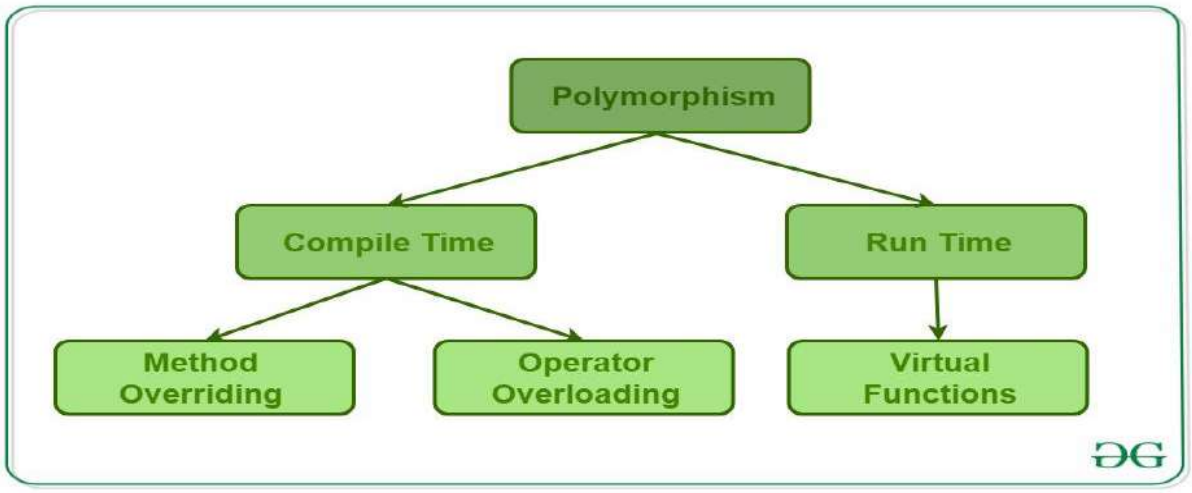
- Polymorphism is a property through which any message can be sent to objects of multiple classes, and every object has the tendency to respond in an appropriate way depending on the class properties.

Polymorphism is the key power of object-oriented programming. It is so important that languages that don't support polymorphism cannot advertise themselves as Object-Oriented languages. Languages that possess classes but have no ability of polymorphism are called object-based languages. Thus it is very vital for an object-oriented programming language.

Types of Polymorphism [click here for video]

In Java, Polymorphism can be divided into two types:

- Compile-time Polymorphism
- Run-time Polymorphism



Compile time polymorphism

It is also known as static polymorphism. The compile-time polymorphism can be achieved through method overloading and operator overloading in Java. The follwing section shows the difference between overloading and overriding.

Method Overloading

In a Java class, we can create methods with the same name if they differ in parameters. For example,

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

This is known as method overloading in Java. Let's take a working example of method overloading.

Example 1: Method Overloading

```
class Demo {
 public void displayPattern(){
   for(int i = 0; i < 10; i++) {
     System.out.print("* ");
   }
 }

 public void displayPattern(char symbol) {
   for(int i = 0; i < 10; i++) {
     System.out.print(symbol+" ");
   }
 }
}

class Main {
 public static void main(String[] args) {
   Demo d1 = new Demo();
   d1.displayPattern();
   System.out.println("\n");
   d1.displayPattern('#');
 }
}
```

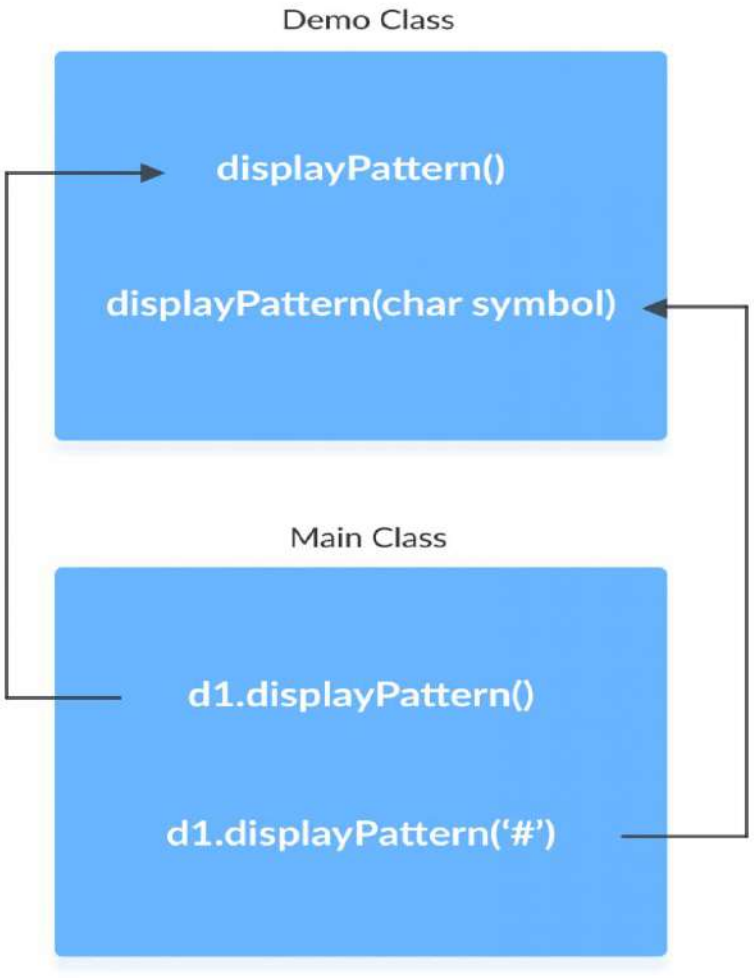
Output:

```
* * * * * * * * * *
# # # # # # # # # #
```

In the above program, the displayPattern() method is overloaded.

- If we call the method without passing any arguments, a pattern of * is created.

- If we call the method by passing a character as an argument, a pattern of that character is created.



Operator Overloading [click here for video]

Some operators in Java behave differently with different operands. For example,

- + operator is overloaded to perform numeric addition as well as string concatenation,
- operators like &, |, and ! are overloaded for logical and bitwise operations.

The + operator in Java is used to perform two specific functions. When it is used with numbers (integers and floating-point numbers), it performs addition. For example,

```
int a = 5;
int b = 6;
int sum = a + b;      //  Output = 11
```

And when we use + operator with strings, it performs string concatenation. For example,

```
String firstName = "abc ";
String lastName = "xyz";
name = firstName + lastName;      //  Output = abc xyz
```

Note: ***In languages like C++, we can define operators to work differently for different operands. However, Java doesn't support user-defined operator overloading.***

Run-time Polymorphism click here for video

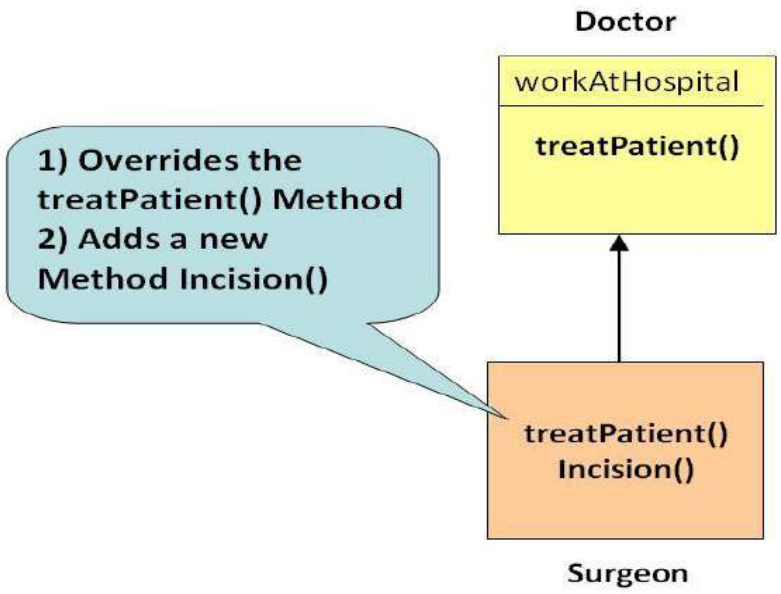
It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Rules for Method Overriding

- The method signature i.e. method name, parameter list and return type have to match exactly.
- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.

Suppose the same method is created in the superclass and its subclasses. In this case, the method that will be called depends upon the object used to call the method.



Example 2: Doctor & Patient click here for video

```
class Doctor{
    public void treatPatient() {
        System.out.println("Doctor treatment..");
        // treatment code goes here
    }
    }
class Surgeon extends Doctor {
        public void treatPatient() {
            System.out.println("Surgeon treatment..");
            // treatment of surgeon goes here
        }
    }
class test{
        public static void main (String args[]){

            Doctor doctorObj = new Doctor();
            // treatPatient method in class Doctor will be executed
            doctorObj.treatPatient();

            Surgeon surgeonObj = new Surgeon();
            // treatPatient  method in class Surgeon  will be executed
            surgeonObj.treatPatient();
        }
    }
```

Output:

```
Doctor treatment..
Surgeon treatment..
```

Difference between Overloading and Overriding

Method Overloading	Method Overriding
Method overloading is in the same class, where more than one method have the same name but different signatures.	Method overriding is when one of the methods in the super class is redefined in the sub-class. In this case, the signature of the method remains the same.

Ex:
`void sum (int a , int b);`
`void sum (int a , int b, int c);`
`void sum (float a, double b);`
	Ex:
	`class X{`
	`  public int sum(){`
	`  // some code`
	`  }`
	`}`
	
	`class Y extends X{`
	`  public int sum(){`
	`  //overridden method`
	`  //signature is same`
	`  }`
	`}`

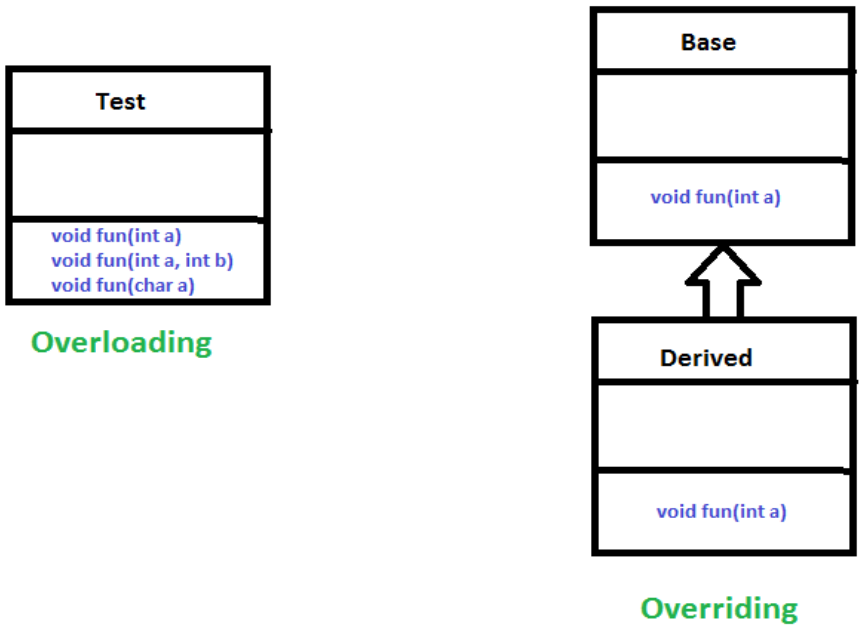


Figure 1 Overloading VS overriding

What is Dynamic Polymorphism? click here for video

Dynamic Polymorphism is the mechanism by which multiple methods can be defined with same name and signature in the super class and subclass and the call to an overridden method are resolved at run time.

Step towards Dynamic Polymorphism

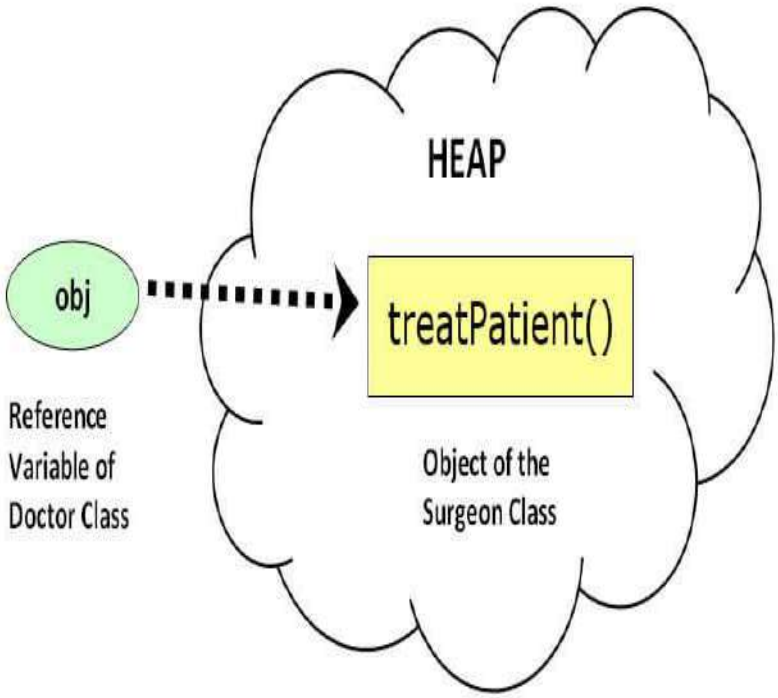
A reference variable of the super class can refer to a sub class object

```
Doctor obj = new Surgeon();
```

Consider the statement

```
obj.treatPatient();
```

Here the reference variable "obj" is of the parent class, but the object it is pointing to is of the child class (as shown in the diagram).



obj.treatPatient() will execute treatPatient() method of the sub-class - Surgeon

If a base class reference is used to call a method, the method to be invoked is decided by the JVM, depending on the object the reference is pointing to.

For example, even though obj is a reference to Doctor, it calls the method of Surgeon, as it points to a Surgeon object

This is decided during run-time and hence termed ***dynamic*** or ***run-time polymorphism***

Super Keyword [click here for video]

What if the treatPatient method in the Surgeon class wants to execute the functionality defined in Doctor class and then perform its own specific functionality?

In this case, keyword super can be used to access methods of the parent class from the child class.
The treatPatient method in the Surgeon class could be written as:

```
treatPatient(){

     super.treatPatient();
     //add code specific to Surgeon
}
```

The keyword super can be used to access any data member or methods of the super class in the sub class.

Another example of method overriding is as follows.

Example 3: Dynamic Polymorphism → Method Overriding click here for video

```
class Animal {
   public void makeSound(){
      System.out.println("Animal sound...");
   }
}

class Dog extends Animal {

   public void makeSound() {
      System.out.println("Dog: Bark bark...");
   }
}

class Cat extends Animal {

   public void makeSound() {
      System.out.println("Cat :Meow meow...");
   }
}
class Main {
   public static void main(String[] args) {
      Dog  d1 = new Dog();
      d1.makeSound();

     Cat c1 = new Cat();
     c1.makeSound();
     System.out.println("\nwhat if we create object of parent: Animal??");
     System.out.println("Do you think the output change??\n");

     Animal a=new Dog();
     a.makeSound();
     a=new Cat();
     a.makeSound();
     System.out.println("Great...This is called dynamic polymorphism
");
```

```
    }
}
```

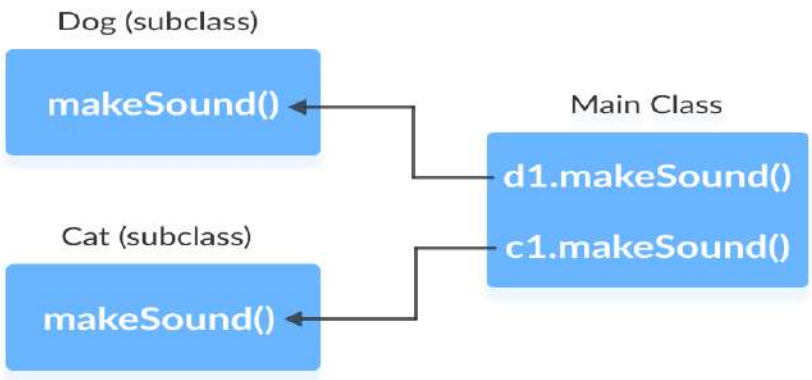
Output:

```
Dog: Bark bark. . .
Cat: Meow-meow. . .

what if we create object of parent: Animal??
Do you think the output change??
Dog: Bark bark...
Cat: Meow meow...
Great...This is called dynamic polymorphism
```

In the above example, the method makeSound() has different implementations in two different classes. When we run the program,

- the expression d1.makeSound() will call the method of Dog class. It is because d1 is an object of the Dog class.
- the expression c1.makeSound() will call the method of Cat class. It is because c1 is an object of the cat class.
- But the call a.makeSound() calls method based on referenced class object

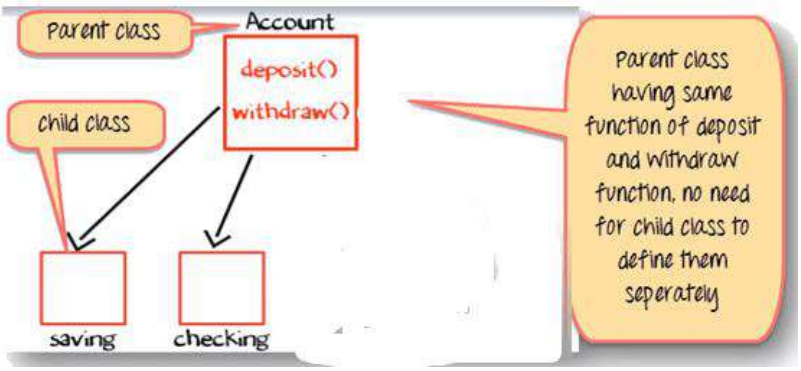


The method that will be called is determined during the execution of the program. Hence, method overriding is a run-time polymorphism.

Another Example Scenario → Polymorphism

We have one parent class, 'Account' with function of deposit and withdraw. Account has 2 child classes

The operation of deposit and withdraw is same for Saving and Checking accounts. So the inherited methods from Account class will work.

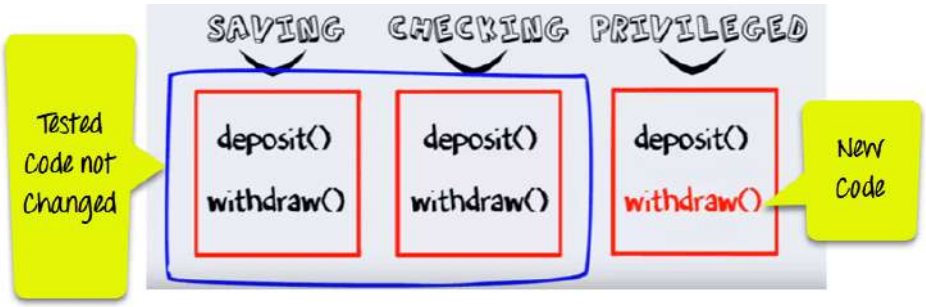


Change in Software Requirement

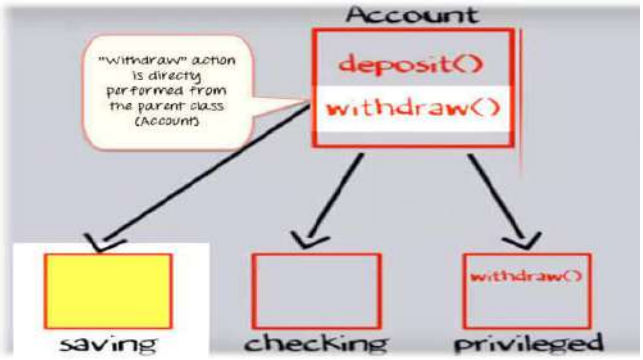
There is a change in the requirement specification, something that is so common in the software industry. You are supposed to add functionality privileged Banking Account with Overdraft Facility.

For a background, overdraft is a facility where you can withdraw an amount more than available the balance in your account.

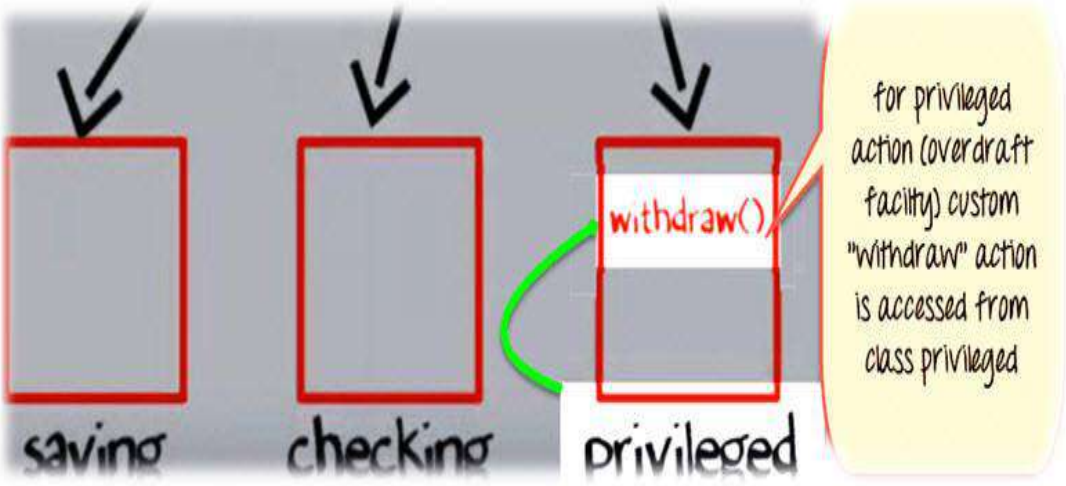
So, withdraw method for privileged needs to implemented afresh. But you do not change the tested piece of code in Savings and Checking account. This is advantage of OOPS



Step 1) Such that when the "withdrawn" method for saving account is called a method from parent account class is executed.



Step 2) But when the "Withdraw" method for the privileged account (overdraft facility) is called withdraw method defined in the privileged class is executed. This is **Polymorphism.**



Difference between Static & Dynamic Polymorphism click here for video

Static Polymorphism	Dynamic Polymorphism
It relates to method overloading.	It relates to method overriding.
Errors, if any, are resolved at compile time. Since the code is not executed during compilation, hence the name static.	In case a reference variable is calling an overridden method, the method to be invoked is determined by the object, your reference variable is pointing to. This is can be only determined at runtime when code in under execution, hence the name dynamic.
Ex:	**Ex:**

```
void sum (int a , int b);
void sum (float a, double b);
int sum (int a, int b); //compiler gives
 error.
```

```
//reference of parent pointing to child
  object
 Doctor obj = new Surgeon();
 // method of child called
 obj.treatPatient();
```

Example #4: Employee & Salary click here for video

```
public class Employee {
   private String name;
   private String address;
   private int number;

   public Employee(String name, String address, int number) {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }

   public void mailCheck() {
      System.out.println("Mailing a check to " + this.name + " " +
this.address);
   }

   public String toString() {
      return name + " " + address + " " + number;
   }

   public String getName() {
      return name;
   }

   public String getAddress() {
      return address;
   }

   public void setAddress(String newAddress) {
      address = newAddress;

   }
    public int getNumber() {
      return number;
   }
}
```

Now suppose we extend Employee class as follows −

```
public class Salary extends Employee {
```

```
   private double salary; // Annual salary

   public Salary(String name, String address, int number, double
salary) {
      super(name, address, number);
      setSalary(salary);
   }

   public void mailCheck() {
      System.out.println("Within mailCheck of Salary class ");
      System.out.println("Mailing check to " + getName()
      + " with salary " + salary);
   }

   public double getSalary() {
      return salary;
   }

   public void setSalary(double newSalary) {
      if(newSalary >= 0.0) {
         salary = newSalary;
      }
   }

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

Now, you study the following program carefully and try to determine its output –

```
public class VirtualDemo {

   public static void main(String [] args) {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3,
3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2,
2400.00);
      System.out.println("Call mailCheck using Salary reference --
");
      s.mailCheck();
```

```
      System.out.println("\n Call mailCheck using Employee
reference--");
      e.mailCheck();
   }
}
```

This will produce the following result –

Output

```
Constructing an Employee
Constructing an Employee

Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

Q: Write code for the following classes.

You are required to implement the above example of “bank” using the concept discussed in this lesson (dynamic polymorphism).

Note: This is not your assignment of week 13, the only thing i want from you to practice the topic for your better understanding.