Example -9 : Polymorphism – overloading and hiding methods of base class And anonymous inner class for overriding bombardment method

Three types of Flying machines are creating from base class. All of them override the fly() and tostring() method. bombarding method is overloaded in Jet call, so it is hiding the base class method.

File : **FlyingMachine.java**

```
// Base class for different type of flying machine
public class FlyingMachine {
   private int noEngine;
   private int noPassenger;
   public void fly() {
      out.println("No implementation");
   }
   public void setEngine(int ne) {
          noEngine = ne;   }
   public int getEngine()    {   return noEngine; }
   public void setPassenger(int np) {
           noPassenger = np;   }
   public int getPassenger()  {
           return noPassenger; }
   public void bombardment(String fmType ) {
       out.println(fmType + " may not bombard
       missile");
   }
}
```

File : **Jet.java**

```
public class Jet extends FlyingMachine {
   @Override
   public void fly() {
      out.println("Jet : Start, taxi, fly with " +
            getEngine() + " Engine(s) and "
            + getPassenger() + " Passenger(s)");
   }
   public void bombardment(int missile) {
      out.println("Jet Can Fire " + missile + "
       missile(s) at a time");
   }
   public String toString() {   return "Jet";   }
}
```

File : **Helicopter .java**

```
public class Helicopter extends FlyingMachine {
   @Override
   public void fly() {
      out.println("Helicopter : Start vertically,
        hover, fly with " + getEngine() + " Engine(s)
        and " + getPassenger() + " Passenger(s)");
   }
   public String toString() {   return "Helicopter";
}
```

File : **PassengerPlane.java**

```
public class PassengerPlane extends
                                FlyingMachine {
   @Override
   public void fly() {
      out.println("PassengerPlane : Start, taxi,
       thrust, lift with " + getEngine() + " Engine(s)
          and " + getPassenger() + " Passenger(s)");
   }
   public String toString() {
              return "PassengerPlane";   }
}
```

File : **MakeThingFly.java**

This class provides the method for flying different flying machine

```
public class MakeThingFly   {

   public void letTheMachinesFly(List<FlyingMachine> flyingMachines) {
      for (FlyingMachine flyingMachine : flyingMachines) {
         flyingMachine.fly();
         if ( flyingMachine.toString() == "Jet")
            //  Downcasting to access the overloaded method in Jet, otherwise we cannot access it
            ((Jet)flyingMachine).bombardment(2);
         else
            flyingMachine.bombardment(flyingMachine.toString());

      }
   }
}
```

File : **OuterFly.java**

```
import static java.lang.System.out;
public class OuterFly {
   /*  This creates an anonymous inner class: */
   FlyingMachine flyIns = new FlyingMachine() {
      @Override
      public void bombardment(String fmType ) {
         out.println(fmType + " will bombard 5 missile at a time");
      }
      // New method may be defined, but we cannot call it using flyIns reference
      public void newMthod(String fmType ) {
         out.println(fmType + " will not support new method");
      }
   };

}
```

Driver snippet

```
private void DriverFlying () {

      //  Anonimous Inner Class
      OuterFly ofly = new OuterFly();
      ofly.flyIns.setEngine(4);
      ofly.flyIns.setPassenger(2);
      ofly.flyIns.bombardment("Fighter Plane");
     // ofly.flyIns.newMethod("Fighter Plane");
```

```
        System.out.println("===========Polymorphism =========");
        List<FlyingMachine> machines = new ArrayList<FlyingMachine>();

        FlyingMachine fm = new FlyingMachine();
        machines.add(fm);
        Jet jt = new Jet();     jt.setPassenger(2);  jt.setEngine(2);
        machines.add(jt);
        Helicopter hel = new Helicopter();         hel.setPassenger(2);   hel.setEngine(1);
        machines.add(hel);
        PassengerPlane pp = new PassengerPlane();   pp.setPassenger(270);  pp.setEngine(2);
        machines.add(pp);

        new MakeThingFly().letTheMachinesFly(machines);
}
```

Example -10: Using new java 8 Date Time API and TemporalAdjusters

- **Java 8 Date Time API** is designed to overcome all the flaws in the legacy date time implementations.
- Java 8 code should use Date-Time classes from the new java.time package and not the earlier java.util based date classes. One of the interfaces included in this new library is ***TemporalAdjuster***.
- We will explain how date and time adjustments can be performed using Java 8's new **TemporalAdjuster** interface.

- A ***Temporal*** (Temporal English meaning is Time-based) object defines a representation of a date, time, or a combination of both, depending on the implementation we're going to be using.
- There are a number of implementations of the *Temporal* interface, including:

    1. *LocalDate* – which represents a date without a timezone
    2. *LocalDateTime* – which represents a date and time without a timezone
    3. *HijrahDate* – which represents a date in the Hijrah calendar system
    4. *MinguoDate* – which represents a date in the Minguo calendar system
    5. *ThaiBuddhistDate* – which represents a date in the Thai Buddhist calendar system

- **TemporalAdjuster** interface is for doing the date mathematics. *TemporalAdjuster* allows us to perform complex date manipulations. For example to get the "Second Saturday of the Month" or "Next Tuesday". Simply put, TemporalAdjuster is a strategy for adjusting a Temporal object. Before getting into the usage of TemporalAdjuster, let's have a look at the Temporal interface itself.

- *TemporalAdjuster* interface has many predefined implementations in the ***TemporalAdjusters*** class. The class **TemporalAdjusters** has many predefined static methods that return a TemporalAdjuster object to adjust **Temporal** objects in many different ways no matter what implementation of Temporal they might be.

- The TemporalAdjuster is a key tool for modifying a temporal object. This interface has a method ***adjustInto****(Temporal)* and it can be directly called by passing the *Temporal* object. It takes input as the temporal value and returns the adjusted value. It can also be invoked using '***with'*** method of the temporal object to be adjusted.
- Java designers have, in fact, thought of the most common of such date adjustment scenarios, and provided predefined TemporalAdjuster implementations via the TemporalAdjusters class. Let us first take a look at TemporalAdjusters and the predefined Temporal Adjusters it provides.

Table: Predefined Temporal Adjusters in **TemporalAdjusters** class

Method Name	Purpose
firstDayOfMonth(), **lastDayOfMonth()**	Get first/last day of month.
firstDayOfNextMonth()	Get first day of next month.
firstDayOfNextYear()	Get first day of next year.
firstDayOfYear(), **lastDayOfYear()**	Get first/last day of year.
firstInMonth(), **lastInMonth()**	Get first/last occurrence of a **DayOfWeek** in month.
next(), **previous()**	Next/Previous occurrence of **DayOfWeek**.
nextOrSame(), **previousOrSame()**	Next/current **DayOfWeek**
dayOfWeekInMonth()	Ordinal **DayOfWeek** in month

6.

- Certain date-time modification operations may need to be applied to dates repeatedly based on business and application requirements. These individual TemporalAdjuster instances can be used to carry out the desired date modifications, or adjustments, as and when required.

- If any of the above date adjustments matches your requirement, then you don't need to implement a new TemporalAdjuster of your own. Simply go ahead and get a reference to the instance of your required date adjuster using the static factory method mentioned above.

File : **DateTimeManipulation.java**

```
import java.time.DayOfWeek;          import java.time.LocalDate;
import java.time.LocalTime;          import java.time.ZoneId;
import java.time.temporal.TemporalAdjusters;
import java.time.format.DateTimeFormatter;          import java.util.Locale;


public class DateTimeManipulation {
   //   Local Time example
   public void LocalTimeExample() {

         //Current Time
         LocalTime curTime = LocalTime.now();        System.out.println("Current Time="+ curTime);
```

LocalTime is an immutable class whose instance represents a time in the human readable format. It's default format is hh:mm:ss.zzz.

```
        //Creating LocalTime by providing input arguments, last parameter in nanosec
        LocalTime specificTime = LocalTime.of(12,20,25,40);
        System.out.println("Specific Time of Day= " + specificTime);

        //  If we try creating time by providing invalid inputs DateTime exception will occur

        //Current date in "Asia/Kolkata", you can get it from ZoneId javadoc
        LocalTime timeKolkata = LocalTime.now(ZoneId.of("Asia/Kolkata"));
        System.out.println("Current Time in IST= " + timeKolkata);

        //  This returns a LocalTime with the specified second-of-day. The nanosecond field will be set
            to zero.
        LocalTime specificSecondTime = LocalTime.ofSecondOfDay(10000);
        System.out.println("10000th second time= " + specificSecondTime);
}

public void DateMath() {

   LocalDate curDate = LocalDate.now();
   System.out.println("Current date : " + curDate);

   //  localDate.with() method is invoked with
       the parameter being the TemporalAdjuster
       instance returned by
      TemporalAdjusters.firstDayOfMonth() method.

   LocalDate fdtMth = curDate.with(TemporalAdjusters.firstDayOfMonth());
   System.out.println("First Day Of Month : " + fdtMth);

   LocalDate ldtMth = curDate.with(TemporalAdjusters.lastDayOfMonth());
   System.out.println("lastDayOfMonth : " + ldtMth);

   LocalDate nxtMon = curDate.with(TemporalAdjusters.next(DayOfWeek.MONDAY));
   System.out.println("Next Monday : " + nxtMon);

   fdtMth = curDate.with(TemporalAdjusters.firstDayOfNextMonth());
   System.out.println("First Day Of NextMonth : " + fdtMth);

   //Get the second saturday of next month
   LocalDate firstInYear = LocalDate.of(curDate.getYear(),curDate.getMonth(), 1);
   LocalDate secondSaturday = firstInYear.with(
            TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY))
            .with(TemporalAdjusters.next(DayOfWeek.SATURDAY));
   System.out.println("Second saturday on : " + secondSaturday);
```

LocalDate is an immutable class that represents Date with default format of yyyy-MM-dd.

- We can use now() method to get the current date.
- We can also provide input arguments for year, month and date to create LocalDate instance.

```
    }
    //  Java 8 – Get Day of Week, Month in Spanish, French for any date using Locale
    //  Show date in French or Spanish using java.util.Locale in Java 8

    public void DayOfWeekWithLocale() {

        //  The java.time.LocalDate.of(int year, Month month, int dayOfMonth) method obtains an
            instance of LocalDate from a year, month and day.

        LocalDate localDate=LocalDate.of(2018,8,01);

        //  Day of week and month in French
        //  To convert the date into French, LocalDate.format() method is used with 2 parameters.
        //  •  The 1st parameter is a java.time.format.DateTimeFormatter instance with format –
               "EEEE, dd MMMM, yyyy".  EEEE specifies the full name of weekday. MMMM specifies the
               full name of the month. dd is 2-digit date and yyyy is 4-digit year.
        //  •  The 2nd parameter specifies the locale using a java.util.Locale instance. Since, the date
               is to be formatted in French, so Locale.FRENCH is used.

        String dateInFrench=localDate.format(DateTimeFormatter.ofPattern("EEEE, dd MMMM,
                                                        yyyy",Locale.FRENCH));
        System.out.println(localDate + " in French: "+ dateInFrench);

        //Day of week and month in Spanish
        Locale spanishLocale=new Locale("es", "ES");
        String dateInSpanish=localDate.format(DateTimeFormatter.ofPattern("EEEE, dd MMMM,
                                        yyyy",spanishLocale));
        System.out.println(localDate + " in Spanish: "+ dateInSpanish);

        //English is the default locale for my system on which JVM is running
        String dateInEnglish = localDate.format(DateTimeFormatter.ofPattern("EEEE, dd MMMM,
                                        yyyy",Locale.getDefault()));
        System.out.println(localDate + " in English(default): "+ dateInEnglish);
     }
}
```

Driver snippet for execution in main()

```
        System.out.println("===========TemporalAdjuster (Related to time)=========");
        // DateTime Manipulation
        DateTimeManipulation dtm = new DateTimeManipulation();
        dtm.LocalTimeExample();
        dtm.DateMath();
        dtm.DayOfWeekWithLocale();
```

Example -11: Exception handling and user defined exception

File : **ExceptionDemo.java**

```
public class ExceptionDemo {

  public void SimpleExcMethod()
  {
    try{
      int num1=30, num2=0;      int output=num1/num2;
      System.out.println ("Result = " +output);
    }
    //  Three way of printing Exception
    //  1.        Print the java.lang.Exception object
    //  2.        Using public void printStackTrace() method of exception method
    //  3.        Using public String getMessage() method of exception object

    catch(ArithmeticException e){
      //e.printStackTrace();
      System.out.println(e);
      System.out.println(e.getMessage());
      System.out.println ("Arithmetic Exception: You can't divide an integer by 0" );
    }
    try{
     int a[]=new int[10];
     //Array has only 10 elements
     a[11] = 9;
    }
    catch(ArrayIndexOutOfBoundsException e){
      System.out.println ("ArrayIndexOutOfBounds");
    }
    try{
          int num=Integer.parseInt ("XYZ") ;
          System.out.println(num);
    }catch(NumberFormatException e){
           System.out.println("Number format exception occurred");
     }
     try{
          String str="easysteps2buildwebsite";
          System.out.println(str.length());;
          char c = str.charAt(0);
          c = str.charAt(40);
          System.out.println(c);
    }catch(StringIndexOutOfBoundsException e){
           System.out.println("StringIndexOutOfBoundsException!!");
```

```
  }

  try{
        String str=null;
        System.out.println (str.length());
   }catch(NullPointerException e){
        System.out.println("NullPointerException..");
   }
}

public void NestedExpMethod()  {
   //  Parent try block
   //  Two try-catch block inside main try block's body marked as block 1 and block 2.
        try {
       //    Child try block1
       try {
          System.out.println("Inside block1");
          int b =150/0;
          System.out.println(b);
       }
       catch(ArithmeticException e1) {
          System.out.println("Exception: e1");
       }
       //Child try block2
       //   Here also ArithmeticException occurred but block 2 catch is only
            ArrayIndexOutOfBoundsException.
       //   So in this case control jump back to Main try-catch(parent) body. Since catch of parent try
            block is handling this exception that's why "Inside parent try catch block" got printed as
            output.
       try {
          System.out.println("Inside block2");
          int b =150/0;
          System.out.println(b);
       }
       catch(ArrayIndexOutOfBoundsException e2) {
          System.out.println("Exception: e2");
       }
       System.out.println("Just other statement");
   }
   catch(ArithmeticException e3){
         System.out.println("Arithmetic Exception");
       System.out.println("Inside parent try catch block");
   }
   catch(ArrayIndexOutOfBoundsException e4){
```

```
            System.out.println("ArrayIndexOutOfBoundsException");
        System.out.println("Inside parent try catch block");
      }
      catch(Exception e5){
            System.out.println("Exception");
        System.out.println("Inside parent try catch block");
      }
      System.out.println("Next statement.");
   }
   //  Since the exception thrown was not handled in the catch blocks the system generated exception
        message  got displayed for that particular exception.
   public void ThrowUnhandledExcMethod()  {
      try{
            char array[] = {'a','b','g','j'};
            System.out.println(array[78]);
      }catch(ArithmeticException e){
            System.out.println("Arithmetic Exception!!");
      }
   }
   //  User Defined Exception
   public void EmployeeAge(int age) throws MyOwnException {
      if(age < 0)
         throw new MyOwnException("Age can't be less than zero");
      else
         System.out.println("Input is valid!!");
   }

   //  Throw an already defined exception using throw keyword

   public int Sum(int num1, int num2){
      if (num1 == 0)
         throw new ArithmeticException("First parameter is not valid");
      else
         System.out.println("Both parameters are correct!!");
      return num1+num2;
   }
}
```

File : **MyOwnException.java**

```
public class MyOwnException extends Exception {
   public MyOwnException(String msg){
     super(msg);
   }
}
```

Driver snippet in main() for execution

```
//   Exception Example
      System.out.println("===========Java Exception =========");
      ExceptionDemo  exDemo = new ExceptionDemo();
      exDemo.SimpleExcMethod();
      exDemo.NestedExpMethod();

      //   User Define Exception
      try {
         exDemo.EmployeeAge(-2);
      }
      catch (MyOwnException e) {
         e.printStackTrace();
      }

      //   Throwing already defined exception
      int res= exDemo.Sum(0,12);
      System.out.println(res);
      System.out.println("Continue Next statements");
```