Metropolitan State University, St. Paul, MN
ICS 372 Object-Oriented Design and Implementation
Class Exercise 3 Solution

1. Implement a collection class named `StudentLinkedList` for storing `Student` objects with the following three methods.

```
public  interface StudentList {
public void add(Student student);
public Student search(String studentId);
public Student delete(String studentId);
}
```

Assume that the class `Student` has a method named `getStudentId()` to return the student id.

Use the `LinkedList` class in java.util to implement the class.

```
import java.util.LinkedList;
import java.util.List;

public class StudentLinkedList implements StudentList {
   private List<Student> students = new LinkedList<Student>();
   @Override
   public void add(Student student) {
      students.add(student);
   }

   @Override
   public Student search(String studentId) {
      for (Student student : students) {
         if (student.getStudentId().contentEquals(studentId)) {
            return student;
         }
      }
      return null;
   }

   @Override
   public Student delete(String studentId) {
      Student student = search(studentId);
      if (student == null) {
         return null;
      }
      students.remove(student);
      return student;
   }
```

```
}
```

The next few questions are based on the following classes.

```
public class Building {
   private String address;
   public Building(String address) {

   }
}

public class CommercialBuilding extends Building {
   private int numberOfClients;

   public CommercialBuilding(String address, int numberOfClients) {


   }

}

public class ResidentialBuilding extends Building {
    private String ownersNames;

    public ResidentialBuilding(String address, String ownersNames) {

    }
}

Object o;
Building b = new Building();
ResidentialBuilding r = new ResidentialBuilding();
CommercialBuilding c = new CommercialBuilding();
```

2. Complete the given constructors for the three classes.

```
public class Building {
   private String address;
   public Building(String address) {
       this.address = address;
   }
}

public class CommercialBuilding extends Building {
    private int numberOfClients;
```

```
    public CommercialBuilding(String address, int numberOfClients) {
       super(address);
       this.numberOfClients = numberOfClients;
   }
}

public class ResidentialBuilding extends Building {
   private String ownersNames;
   public ResidentialBuilding(String address, String ownersNames) {
      super(address);
      this.ownersNames = ownersNames;
   }
}
```

3. Suppose the cost of a building is to be computed and stored. The way the cost is computed is different for commercial buildings and residential buildings. The cost also has to be made available to code external to the three classes. How would you redesign the three classes?

```
public abstract class Building {
   private String address;
   protected double cost;

   public Building(String address) {
      this.address = address;
   }

   public double getCost() {
      return cost;
   }

   public abstract void computeCost();
}

public class CommercialBuilding extends Building {
   private int numberOfClients;

   public CommercialBuilding(String address, int numberOfClients) {
      super(address);
      this.numberOfClients = numberOfClients;
   }

   public void computeCost() {
   // TODO
   }
}

public class ResidentialBuilding extends Building {
```

```
   private String ownersNames;

   public ResidentialBuilding(String address, String ownersNames) {
      super(address);
      this.ownersNames = ownersNames;
   }

   public void computeCost() {
   // TODO
   }
}
```

4. Predict the output of executing the following code.

```
public class B {
   public void m1(B b1) {
      System.out.println("B m1");
   }

   public void m2(B b1, B b2) {
      System.out.println("B m2");
      b1.m1(b1);
      b2.m1(b1);
   }
}

public class D extends B {
   public void m1(B b1) {
      System.out.println("D m1");
   }
}

public class E extends B {
   public void m1(B b1) {
      System.out.println("E m1");
   }

   public void m2(B b1, B b2) {
      System.out.println("E m2");
      super.m2(b1, b2);
   }
}

public class Driver {
   public static void main(String[] s) {
      B[] bList = { new B(), new D(), new E(), new D() };
      System.out.println("Call 1");
      bList[0].m2(bList[0], bList[1]);
```

```
      System.out.println("Call 2");
      bList[1].m2(bList[0], bList[2]);
      System.out.println("Call 3");
      bList[2].m2(bList[1], bList[2]);
      System.out.println("Call 4 ");
      bList[2].m2(bList[2], bList[3]);
   }
}
```

```
Call 1
B m2
B m1
D m1
Call 2
B m2
B m1
E m1
Call 3
E m2
B m2
D m1
E m1
Call 4
E m2
B m2
E m1
D m1
```

5. Complete the code to serialize the data created in the `SerializationClass` class.

```
public interface Entity {
   public String getName();
}

public interface EntityList {
   public void add(Entity entity);
}

public class Building implements Entity {
   private String name;
   private EmployeeList employees;

   public Building(String name) {
      this.name = name;
      employees = new EmployeeList();
   }
```

```
   public void addEmployee(Employee employee) {
      employees.add(employee);
   }

   public void print() {
      System.out.println("Building " + name + " " + employees);
   }
}

import java.util.LinkedList;
import java.util.List;

public class BuildingList implements EntityList {
   private List<Building> buildings = new LinkedList<Building>();

   public void add(Entity building) {
      buildings.add((Building) building);
   }
}

public class Employee implements Entity {
   private String name;

   public Employee(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
}

public class EmployeeList implements EntityList {
   private List<Employee> employees = new LinkedList<Employee>();

   public void add(Entity entity) {
      employees.add((Employee) entity);
   }
}

import java.io.Serializable;
public interface Entity extends Serializable {
   public String getName();
}

import java.io.Serializable;
public interface EntityList extends Serializable {
   public void add(Entity entity);
```

```
}

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializationClass {
    public static void main(String[] args) {
      BuildingList buildingList = new BuildingList();
      Building building1 = new Building("B1");
      EmployeeList employeeList = new EmployeeList();
      Employee employee1 = new Employee("E1");
      employeeList.add(employee1);
      building1.addEmployee(employee1);
      buildingList.add(building1);
      FileOutputStream fos = null;
      try {
         fos = new FileOutputStream("data");
      } catch (FileNotFoundException fnfe) {
         fnfe.printStackTrace();
         System.exit(0);
     }
     ObjectOutputStream oos = null;
     try {
           oos = new ObjectOutputStream(fos);
           oos.writeObject(buildingList);
      } catch (IOException ioe) {
           ioe.printStackTrace();
           System.exit(0);
      }
   }
}
```

6. Complete the code to deserialize the data stored in the `SerializationClass` class of Question 5.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeSerializationClass {
   public static void main(String[] args) {
      BuildingList buildingList;
      FileInputStream fis = null;
      try {
         fis = new FileInputStream("data");
```

```
      } catch (FileNotFoundException fnfe) {
         fnfe.printStackTrace();
         System.exit(0);
      }
      ObjectInputStream ois = null;
      try {
         ois = new ObjectInputStream(fis);
         buildingList = (BuildingList) ois.readObject();
      } catch (IOException ioe) {
         ioe.printStackTrace();
     System.exit(0);
      } catch (ClassNotFoundException cnfe) {
          cnfe.printStackTrace();
          System.exit(0);
      }
    }
}
```