CSE1IOO/CSE4IOO Sample Written Exam
Sample Solution

Question 1

(a)
```
public abstract class Animal
{
   private String id;
   private char code;

   public Animal(String id, char code)
   {
      this.id = id;
      this.code = code;
   }

   public String getId()
   {
      return id;
   }

   public char getCode()
   {
      return code;
   }

   public abstract double getFoodAmount();
}
```

(b)
```
public class Cow extends Animal
{
   private double weight;

   public Cow(String id, char code, double weight)
   {
      super(id, code);
      this.weight = weight;
   }

   public double getWeight()
   {
      return weight;
   }
   public double getFoodAmount()
   {
      if (getCode() == 'M')
      {
         return .05 * weight;
      }
      else
      {
         return .02 * weight;
      }
   }

   public String toString()
   {
      return "Cow[id: " + getId() + ", code: " + getCode()
         + ", weigth: " + weight + "]";
   }
}
```

(c)
```
public class Sheep extends Animal
{
   private int age;

   public Sheep(String id, char code, int age)
   {
      super(id, code);
      this.age = age;
   }

   public double getFoodAmount()
   {
      if (age < 8)
      {
         return 1;
      }
      else if (age < 16)
      {
         return 1.5;
      }
      else
      {
         return 2;
      }
   }

    public String toString()
    {
      return "Sheep[id: " + getId() + ", code: " + getCode()
         + ", age: " + age + "]";
   }
}
```

(d)
```
public static ArrayList<Animal> readData() throws Exception
{
   Scanner in = new Scanner(new File("CowsSheep.txt"));
   ArrayList<Animal> list = new ArrayList<Animal>();
   while(in.hasNext())
   {
      String line = in.nextLine();
      String[] tokens = line.split("\\s");

      if(tokens[1].equals("Cow"))
      {
         Animal cow = new Cow(tokens[0], tokens[2].charAt(0),
            Double.parseDouble(tokens[3]));
         list.add(cow);
      }
      else
      {
         Animal sheep = new Sheep(tokens[0], tokens[2].charAt(0),
            Integer.parseInt(tokens[3]));
         list.add(sheep);
      }
   }

   return list;
}
```

Question 2

(a)
```
public class InvalidDataException extends Exception
{
   public InvalidDataException()
   {
      super();
   }

   public InvalidDataException(String message)
   {
      super(message);
   }
}
```

(b)
```
   public static double calculateInsurancePremium(
      double carValue, int driverAge) throws Exception
   {

      if(carValue <10000)
      {
         throw new InvalidDataException("ERROR: Car value is outside the valid
              range!");
      }

      if(driverAge < 18 || driverAge > 90)
      {
         throw new InvalidDataException("ERROR: Driver's age is outside the
             valid range!");
      }

      double amount = carValue * 0.05;

      if(driverAge < 21)
      {
         amount = carValue * 0.15;
      }

      return amount;
   }
```

Question 3

```
public static void makeListing(String dirName) throws Exception
{
     File dir = new File(dirName);

     if (dir.isFile())
     {
        System.out.println("There is a file with the name "  + dirName);
     }
     else if(! dir.isDirectory())
     {
        System.out.println("No directory with the name " + dirName);
     }
     else
     {
        PrintWriter out = new PrintWriter(new File("listing.txt"));

        File [] files = dir.listFiles();

        for(File each: files)
        {
           out.println(each.getName());
        }

        out.close();
  }
}
```

Question 4

(a)
```
public interface HazardRating
{
   public abstract double getRating();
}
```

(b)
```
import java.io.Serializable; // this statement is not marked

public class Chemical implements HazardRating, Serializable
{
   private int temperature;
   private double volume;

   public Chemical(int temperature, double volume)
   {
      this.temperature = temperature;
      this.volume = volume;
   }
   public String toString()
   {
       return getClass().getName() +
             "[temperature: " + temperature +
             " volume: " + volume + "]";
   }

   public double getRating()
   {
      return temperature / volume;
   }
}
```

(c)
```
   public static void displayChemicals(ArrayList<HazardRating> list)
   {
      for(HazardRating item: list)
      {
         if(item instanceof Chemical)
         {
            System.out.println(item);
         }
      }
   }
```

(d)

```
public static void save(ArrayList<HazardRating> list, ObjectOutputStream
    out)
{
   try
   {
      for(HazardRating item: list)
      {
         out.writeObject(item);
      }
   }
   catch(Exception e)
   {
      System.out.println(e);
   }

   // Use try/catch block because the writeObject method can throw
       IOException
}
```

NOTE: Actually it is better for the method to have the following header, which indicates that it can throw an IOException:

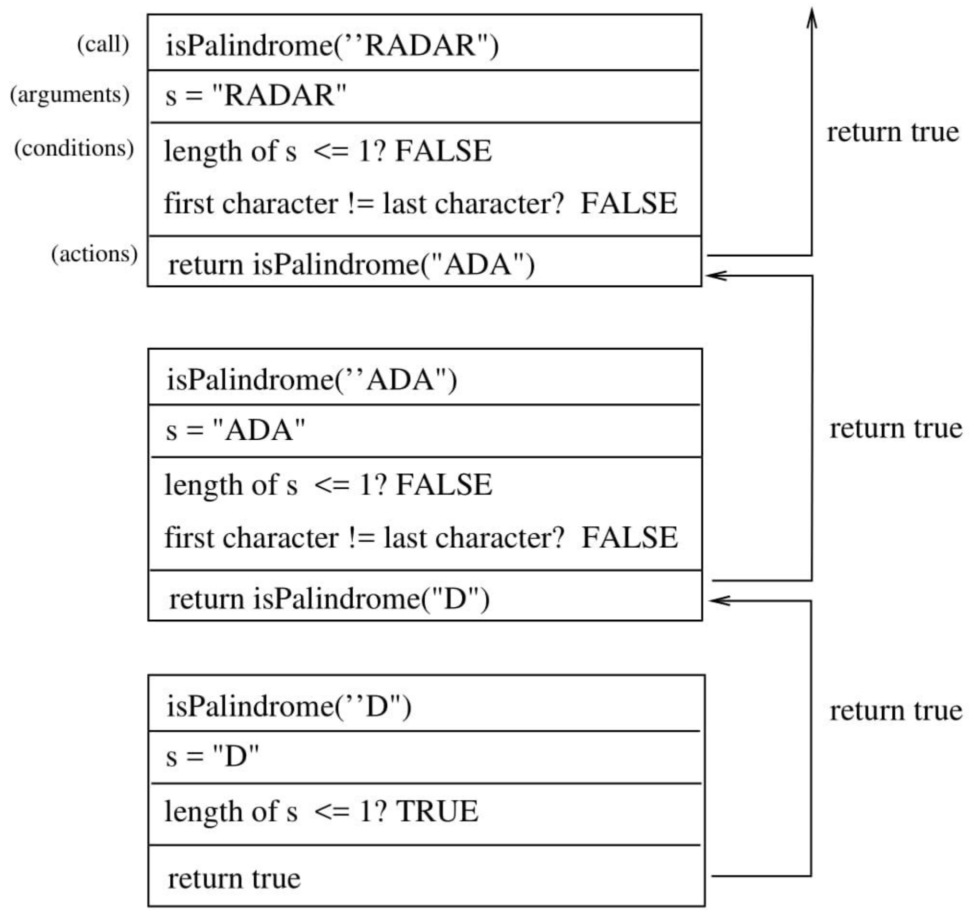
```
public static void save(ArrayList<HazardRating> list, ObjectOutputStream out)
throws IOException // or just Exception
```

Question 5

(a)
```
public static boolean isPalindrome(String s)
{
   if(s.length() <= 1)
   {
      return true;
   }
   else if(s.charAt(0) != s.charAt(s.length() -1))
   // first character is different from last character
   {
      return false;
   }
   else
   {
      return isPalindrome(s.substring(1, s.length() - 1));
   }

}
```

(b) *(We represent each call by a box, and inside the box we record information about the call. For example, we can do this as shown in the diagram below.)*



Question 6

(a)
```
public void addLightToStart(Light newLight)
{
   if (head == null)
   {
      head = newLight;
   }
   else
   {
      newLight.setNextLight(head);
      head = newLight;
   }
}
```

(b)
```
public void turnOnLightsOfColour(String colour)
{

   Light p = head;
   while (true)
   {
      if (p == null)
      {
         break;
      }
      else
      {
         if (p.getColour().equals(colour))
         {
            p.switchOn();
         }

         p = p.getNextLight();

      }
   }
}
```

Question 7

```
public static <E extends Comparable<E>> ArrayList<E> getRange(List<E> list)
{
   // assume that the list has at least one element

   E min = list.get(0);
   E max = list.get(0);

   for(int i = 1; i < list.size(); i++)
   {
      if(list.get(i).compareTo(min) < 0)
      {
         min = list.get(i);
      }

      if(list.get(i).compareTo(max) > 0)
      {
         max = list.get(i);
      }
   }

   ArrayList<E> result = new ArrayList<E>();
   result.add(min);
   result.add(max);

   return result;
}
```

NOTE: For simplicity, we assume that the list has at least one element. If we do not assume this, we need to assume what to return if the list is empty. In the later case, the most natural choice would be to return two null objects for the minimum and maximum.

■