**Name:____Solutions______**

# Exam #1 [Practice]
**15-111/Kesden/Fall 2003**

1. Please consider the following `Person` class. Notice the use of the Java keyword "this". What is "this"? And why is it used in the constructor below?

```
class Person {
  private String fullName;
  private String fullAddress;

  public Person (String fullName, fullAddress) {
    this.fullName = fullName;
    this.fullAddress = fullAddress;
  }

  public String getFullname() {
    return fullName;
  }

  public String getFullAddress() {
    return fullAddress;
  }
}
```

*This is a constant reference to the object that executing the constructor or method. In some sense, it is somewhat like the pronoun "me" or "I". It is used here because the default Java scope is the most local scope – fullName or fullAddress, without an explicit scope, are the variables within the formal argument list (parameters). Remember, the "." is the scope operator. "this." names the method or variable within the current object.*

2. Below is the skeleton of a `PersonFinder` class based on the `Person` class defined above. Please complete this skeleton using a `Vector` as the primary data structure.

```
class PersonFinder {

  private Vector collection;

  // Constructor
  public PersonFinder () {

    collection = new Vector();

  }

  // Return a Person’s fullAddress
  public String findAddress(String fullName) {

     for (int index=0; index< collection.size(); index++)
     {
       Person possibleMatch =
            (Person) collection.elementAt(index);

       if (p.getFullName().equals(fullName))
        return p.getFullAddress();
     }

     return null;
  }


  // Add a Person to this PersonFinder
  public void addPerson (Person newEntry)
  {
     v.add(newEntry);
  }
}
```

3. Below is a skeleton for a `miniVector` class, please complete it. This class should use an array as its primary data structure. Consistent with the semantics of Java's `Vector,` it should grow as needed.

```
public class miniVector {

   private unsigned int nextAvail = 0;
   private Object[] collection = new Object[10];

  public void setElementAt (Object o, int index)  throws ArrayIndexOutOfBoundsException
  {
    collection[index] = o;
  }

  public Object elementAt (int index)  throws ArrayIndexOutOfBoundsException
  {
    return collection[index];
  }

  public void addElement (Object o)
  {
     if (nextAvail >= collection.length)
       grow();

      collection[nextAvail++] = o;
  }

  public void insertElementAt (Object o, int index) throws ArrayIndexOutOfBoundsException
  {
     if (nextAvail >= collection.length)
       grow();

     for (int posn=nextAvail; posn !=index; posn--)
       collection[posn] = collection[posn-1];

     collection[index] = o;
     nextAvail++;
  }

  public void removeElementAt (int index) throws ArrayIndexOutOfBoundsException
  {
     for (int posn=index; posn < nextAvail-1; posn++)
       collection[posn] = collection[posn+1];

     collection[--nextAvail] = null;
  }
```

Question #3, cont.

```
  private void grow()
  {
    Object[] newCollection = new Object[2*collection.length];

    for (int index=0; index< collection.length; collection++)
       newCollection[index] = collection[index];

    collection = newCollection;
  }
}
```

4. Repeat the exercise described for question #3, but this time, please use Java's `LinkedList` as the underlying data structure. Additionally, please create your own `Exception`, `MiniVectorBoundsException` and throw it in the event of an underflow or an overflow (negative index, or index greater than actually exists).

```
public class miniVector {

    public class MiniVectorBoundsException extends Exception
    {
       public MiniVectorBoundsException (String s)
       {
         super (s);
       }
    }

     private LinkedList collection = new LinkedList;


  public void setElementAt (Object o, int index)  throws MiniVectorBoundsException
  {

    if ((index > collection.size()-1) || (index < 0))
      throw new MiniVectorBoundsException
           ("Index out of bounds in setElementAt()");

        collection.set (index, o);
  }
```

Question #4, cont.

```
public Object elementAt (int index)  throws MiniVectorBoundsException
 {
  if ((index > collection.size()-1) || (index < 0))
    throw new MiniVectorBoundsException
         ("Index out of bounds in elementAt()");

 return collection.get(index);
 }

 public void addElement (Object o)
 {
  collection.add(o);
 }

 public void insertElementAt (Object o, int index) throws MiniVectorBoundsException
 {
  if ((index > collection.size()-1) || (index < 0))
    throw new MiniVectorBoundsException
         ("Index out of bounds in insertElementAt()");


  collection.add(index, o);

 }

 public void removeElementAt (int index) throws MiniVectorBoundsException
 {
  if ((index > collection.size()-1) || (index < 0))
      throw new MiniVectorBoundsException
           ("Index out of bounds in removeElementAt()");


  collection.remove(index);

 }
}
```

5. Inheritance is a mechanism that is used to define classes with a certain relationship to each other. Please characterize this relationship. If it is easier for you, please answer the question, "When is it appropriate to define a new class using inheritance?"

*Inheritance is a mechanism used to implement sub-classing. It is appropriate to use inheritance to implement classes that are related by the "is a" relationship.*

6. What is the difference between an object and a class?

*A class is a specification. It defines the properties of a type of object. The object, itself, is a model of something, be it tangible or abstract, that satisfies the definition of the class. Some people like to explain, "A class is to an object what a blueprint is to a house". Many houses might be built according to the same blueprint – each is the same type of house. But, each house lives its own life, keeping its own family warm, dry, safe, &c.*

7. Consider the following definition, what does "p" represent?

```
Person p;
```

*"p" names a variable capable of representing the name of a Person object. "p" is just a primitive variable, it is not an instance of Person class, nor are reference variables, themselves, classes.*

# LinkedList reference
## (Abbreviated Method Summary from Sun’s API Documention):

| |
|---|
| |
| |

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

| | |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |