CMP-338 Solutions Midterm Spring 2017

Version 1

Instructions

- **Write your name and version number on the top of the yellow paper.**
- **Answer all questions on the yellow paper.**
- **One question per page.**
- **Use only one side of the yellow paper.**

1. (16 Points) Multiple Choice:

A. (2 Points) If x is a variable of type int, what is the largest possible value of the expression (x % 5) ?

```
a. 1
b. 4
c. 5
d. 2^31-1
```

B. (2 Points) How many constructors can a class have?

```
a. Exactly one
b. At least one, but no more than
   three
c. Exactly the same as the number
   of data members
d. There is no restriction on the
   number of constructors
```

C. (2 Points) In the following list:
John, Kate, Fred, Mark, Jon, Adam, Drew.
Which element is the head of the list?

```
a. John
b. Mark
c. Drew
d. Adam
```

D. (2 Points) The definition of a subclass includes a(n) _____ clause to indicate its superclass.

```
a. super
b. extends
c. this
d. implements
```

E. (2 Points) Which of the following statements deletes the first node of a linear linked list that has 10 nodes?

```
a. head.setNext(cur.getNext());
b. prev.setNext(cur.getNext());
c. head = head.getNext();
d. head = null;
```

F. (2 Points) If the array: {6, 2, 7, 13, 5, 4} is added to a stack, in the order given, which number will be the first number to be removed from the stack?

```
a. 6
b. 2
c. 5
d. 4
```

G. (2 Points) last-in, first-out (LIFO) property is found in the ADT _____.

```
a. list
b. stack
c. queue
d. tree
```

H. (2 Points) The `enqueue` operation of the ADT queue is similar to the _____ operation of the ADT stack.

```
a. isEmpty
b. peek
c. push
d. pop
```

Version 1

2. (20 Points) Given the following QueueInterface:

```
public interface QueueInterface {
    public void enqueue(Object obj);
    public Object dequeue();
    public Object peek();
}
```

And given the following LinkedQueue that implements QueueInterface as a doubly-linked queue:

```
public class Node {
   private Object object;
   private Node next;
   private Node previous;

   public Node(Object object) {
      this.object = object;
      this.next = null;
      this.previous = null;
   }

   public Object getObject() {
      return this.object;
   }

   public Node getNext() {
      return this.next;
   }

   public void setNext(Node next) {
      this.next = next;
   }

   public Node getPrevious() {
      return this.previous;
   }

   public void setPrevious(Node previous) {
      this.previous = previous;
   }
}
```

```
public class LinkedQueue implements QueueInterface {

   private Node front = null, back = null;

   @Override
   public void enqueue(Object obj) {
      Node newNode = new Node(obj);
      if (this.back == null) {
         this.front = newNode;
         this.back = this.front;
      } else {
         this.back.setNext(newNode);
         newNode.setPrevious(this.back);
         this.back = newNode;
      }
   }

   @Override
   public Object dequeue() {
      Object obj = null;
      if (this.front != null) {
         obj = this.front.getObject();
         this.front = this.front.getNext();
      }

      if (front != null) {
         this.front.setPrevious(null);
      } else {
         this.back = null;
      }
      return obj;
   }

   @Override
   public Object peek() {
      Object obj = null;
      if (this.front != null) {
         obj = this.front.getObject();
      }
      return obj;
   }
}
```

Version 1

3. (40 Points) The correct ArrayStack Implementation is:

```
import java.util.Vector;

public class ArrayStack implements StackInterface {

   private Vector<Object> stackVector = new Vector<Object>();

   private final int INVALID_STACK_POINTER = -1;
   private int stackPointer = INVALID_STACK_POINTER;

   @Override
   public boolean isEmpty() {
      return stackVector.isEmpty();
   }

   @Override
   public int size() {
      return stackVector.size();
   }

   @Override
   public void push(Object obj) {
      stackVector.add(++stackPointer, obj);
   }

   @Override
   public Object pop() {
      Object obj = null;
      if (stackPointer != INVALID_STACK_POINTER) {
         obj = stackVector.elementAt(stackPointer);
         stackVector.removeElementAt(stackPointer--);
      }
      return obj;
   }

   @Override
   public boolean equals(Object oStack) {
      boolean answer = false;
      ArrayStack otherStack;

      if (oStack instanceof ArrayStack) {
         otherStack = (ArrayStack) oStack;
      } else {
         return answer;
      }

      answer = stackVector.equals(otherStack.peekAll());
      return answer;
   }

   @Override
   public Vector<Object> peekAll() {
      return (Vector<Object>)stackVector.clone();
   }
}
```

Version 1

4. (30 Points) Given the following SortInterface:

```java
public interface SortInterface {
   public void sort(Integer[] arrayToSort);
}
```

Write the BubbleSort class that implements the given SortInterface using the Bubble Sort Algorithm.

```java
public class BubbleSort implements SortInterface {

   @Override
   public void sort(Integer[] arrayToSort) {
      int last = arrayToSort.length - 1;

      while(last > 0) {
         int lastSwap = 0;
         int i = 0;

         while (i < last) {
            if (arrayToSort[i].compareTo((Integer)arrayToSort[i+1]) > 0) {
               lastSwap = i;
               Integer temp = arrayToSort[i];
               arrayToSort[i] = arrayToSort[i+1];
               arrayToSort[i+1] = temp;
            }
            i++;
         }
         last = lastSwap;
      }
   }

}
```

Version 2

Instructions

- **Write your name and version number on the top of the yellow paper.**
- **Answer all questions on the yellow paper.**
- **One question per page.**
- **Use only one side of the yellow paper.**

1. (16 Points) Multiple Choice:

A. (2 Points) If s1 is of type String, what does s1.compareTo(s1) return?

a. zero
b. true
c. false
d. Cannot be determined without knowing the value of s1

B. (2 Points) What does it mean for the return type of a method to be void?

a. The method will never return a value
b. The method will return the value zero
c. The method does not take parameters
d. The method does not have a body

C. (2 Points) In the following list:
John, Kate, Fred, Mark, Jon, Adam, Drew.
Which element is the tail of the list?

a. John
b. Mark
c. Drew
d. Adam

D. (2 Points) In Java, a class can extend _____.

a. At most 1 class
b. At most 16 classes
c. At most 32 classes
d. As many classes as required

E. (2 Points) Which of the following statements deletes the node that ***cur*** references?

a. prev.setNext(cur);
b. cur.setNext(prev);
c. cur.setNext(cur.getNext());
d. prev.setNext(cur.getNext());

F. (2 Points) If the array: {6, 21, 35, 3, 6, 2, 13} is added to a stack, in the order given, which of the following is the top of the stack?

a. 2
b. 6
c. 3
d. 13
e. 35

G. (2 Points) first-in, first-out (FIFO) property is found in the ADT _____.

e. list
f. stack
g. queue
h. tree

H. (2 Points) The `pop` operation of the ADT stack is similar to the _____ operation of the ADT queue.

a. isEmpty
b. enqueue
c. dequeue
a. peek

Version 2

2. (20 Points) Given the following StackInterface:

```
public interface StackInterface {
    public void push(Object obj);
    public Object pop();
    public Object peek();
}
```

And given the following LinkedStack that implements StackInterface as a doubly-linked stack:

```
public class Node {
   private Object object;
   private Node next;
   private Node previous;

   public Node(Object object) {
      this.object = object;
      this.next = null;
      this.previous = null;
   }

   public Object getObject() {
      return this.object;
   }

   public Node getNext() {
      return this.next;
   }

   public void setNext(Node next) {
      this.next = next;
   }

   public Node getPrevious() {
      return this.previous;
   }

   public void setPrevious(Node previous) {
      this.previous = previous;
   }
}
```

```
public class LinkedStack implements StackInterface {

   private Node top = null;

   @Override
   public void push(Object obj) {
      Node newNode = new Node(obj);
      if (this.top == null) {
         this.top = newNode;
      } else {
         newNode.setNext(this.top);
         this.top.setPrevious(newNode);
         this.top = newNode;
      }
   }

   @Override
   public Object pop() {
      Object obj = null;
      if (this.top != null) {
         obj = this.top.getObject();
         top = this.top.getNext();
      }

      if (this.top != null) {
         this.top.setPrevious(null);
      }

      return obj;
   }

   @Override
   public Object peek() {
      Object obj = null;
      if (this.top != null) {
         obj = this.top.getObject();
      }
      return obj;
   }
}
```

Version 2

3. (40 Points) The correct ArrayQueue implementation is:

```
import java.util.Vector;

public class ArrayQueue implements QueueInterface {

    private Vector<Object> queueVector = new Vector<Object>();

    @Override
    public boolean isEmpty() {
            return queueVector.isEmpty();
    }

    @Override
    public int size() {
            return queueVector.size();
    }

    @Override
    public void enqueue(Object obj) {
            queueVector.addElement(obj);
    }

    @Override
    public Object dequeue() {
            Object obj = null;
            if (queueVector.size() > 0) {
                    obj = queueVector.elementAt(0);
                    queueVector.remove(0);
            }
            return obj;
    }

    @Override
    public boolean equals(Object oQueue) {
            boolean answer = false;
            ArrayQueue otherQueue;

            if (oQueue instanceof ArrayQueue) {
                    otherQueue = (ArrayQueue) oQueue;
            } else {
                    return answer;
            }

            answer = queueVector.equals(otherQueue.peekAll());

            return answer;
    }

    @Override
    public Vector<Object> peekAll() {
            return (Vector<Object>) queueVector.clone();
    }
}
```

Version 2

4. (30 Points) Given the following SortInterface:

```
public interface SortInterface {
   public void sort(Integer[] arrayToSort);
}
```

Write the InsertionSort class that implements the given SortInterface using the Insertion Sort Algorithm.

```
public class InsertionSort implements SortInterface {

   @Override
   public void sort(Integer[] arrayToSort) {
      int j;
      Integer temp;

      for ( int i = 1 ; i < arrayToSort.length ; i++ ) {
         temp = arrayToSort[i];
         j = i;

         while ((j > 0) && (arrayToSort[j-1].compareTo(temp) > 0)) {
            arrayToSort[j] = arrayToSort[j-1];
            j--;
         }
         arrayToSort[j] = temp;
      }
   }

}
```

Version 3

Instructions

- **Write your name and version number on the top of the yellow paper.**
- **Answer all questions on the yellow paper.**
- **One question per page.**
- **Use only one side of the yellow paper.**

1. (16 Points) Multiple Choice:

A. (2 Points) If we wanted to write an if-statement that executes whenever the real number x is between 10.0 and 20.0, how should the test condition be written?

```
a. 10.0 < x || x > 20.0
b. 10.0 < x && x > 20.0
c. 10.0 < x && x < 20.0
d. 10.0 < x || x < 20.0
```

B. (2 Points) The communication mechanisms among modules are called _____.

```
a. algorithms
b. solutions
c. prototypes
d. interfaces
```

C. (2 Points) In a sorted array, the kth smallest item is given by _____.

```
a. anArray[k-1]
b. anArray[k]
c. anArray[SIZE-k]
d. anArray[SIZE+k]
```

D. (2 Points) In the ADT list, when an item is inserted into position i of the list, _____.

```
a. the position of all items is
   increased by 1
b. the position of each item that
   was at a position smaller than i
   is increased by 1
c. the position of each item that
   was at a position greater than i
   is increased by 1
d. the position of each item that
   was at a position smaller than i
   is decreased by 1 while the
   position of each item that was
   at a position greater than i is
   increased by 1
```

E. (2 Points) Which of the following statements deletes the first node of a linear linked list that has 10 nodes?

```
a. head.setNext(curr.getNext());
b. prev.setNext(curr.getNext());
c. head = head.getNext();
d. head = null;
```

F. (2 Points) If the string w is a palindrome, which of the following is true?

```
a. w minus its first character is a
   palindrome
b. w minus its last character is a
   palindrome
c. w minus its first and last
   characters is a palindrome
d. the first half of w is a
   palindrome
e. the second half of w is a
   palindrome
```

G. (2 Points) If the array: {6, 2, 7, 13, 5, 4} is added to a queue, in the order given, which number will be the first number to be removed from the queue?

```
a. 6
b. 2
c. 5
d. 4
```

H. (2 Points) Operations on a queue can be carried out at _____.

```
a. its front only
b. its back only
c. both its front and back
d. any position in the queue
```

Version 3

2. (20 Points) Given the following QueueInterface:

```
public interface QueueInterface {
    public void enqueue(Object obj);
    public Object dequeue();
    public Object peek();
}
```

And given the following array-based ArrayQueue that implements QueueInterface:

```
import java.util.Vector;

public class ArrayQueue implements QueueInterface {

   private Vector<Object> queueVector = new Vector<Object>();

   @Override
   public void enqueue(Object obj) {
      queueVector.addElement(obj);
   }

   @Override
   public Object dequeue() {
      Object obj = null;
      if (queueVector.size() > 0) {
         obj = queueVector.elementAt(0);
         queueVector.remove(0);
      }
      return obj;
   }

   @Override
   public Object peek() {
      Object obj = null;
      if (queueVector.size() > 0) {
         obj = queueVector.elementAt(0);
      }
      return obj;
   }
}
```

Version 3

3. (40 Points) The correct LinkedStack implementation is:

```java
import java.util.Vector;

public class LinkedStack implements StackInterface {

    private Node stackPtr = null;
    int size = 0;

    @Override
    public boolean isEmpty() {
       return (stackPtr == null);
    }

    @Override
    public int size() {
       return this.size;
    }

    @Override
    public void push(Object obj) {
       Node newNode = new Node(obj);
       if (stackPtr == null) {
          stackPtr = newNode;
       } else {
          newNode.setNext(stackPtr);
          stackPtr = newNode;
       }
       this.size++;
    }

    @Override
    public Object pop() {
       Object obj = null;
       if (stackPtr != null) {
          obj = stackPtr.getObject();
          stackPtr = stackPtr.getNext();
       }
       this.size--;
       return obj;
    }

    @Override
    public boolean equals(Object oStack) {
       boolean answer = false;
       LinkedStack otherStack;

       if (oStack instanceof LinkedStack) {
          otherStack = (LinkedStack) oStack;
       } else {
          return answer;
       }

       Vector<Object> myPV = this.peekAll();
       answer = myPV.equals(otherStack.peekAll());

       return answer;
    }

    @Override
    public Vector<Object> peekAll() {
       Vector<Object> pv = new Vector<Object>();
       Node curNode = this.stackPtr;

       while (curNode != null) {
          pv.add(curNode.getObject());
          curNode = curNode.getNext();
       }

       return pv;
    }
}
```

4. (30 Points) Given the following SortInterface:

```
public interface SortInterface {
   public void sort(Integer[] arrayToSort);
}
```

Write the SelectionSort class that implements the given SortInterface using the Selection Sort Algorithm.

```
public class SelectionSort implements SortInterface {

   @Override
   public void sort(Integer[] arrayToSort) {
      int cur, min;

      for ( cur = 0 ; cur < arrayToSort.length ; cur++ ) {
         min = cur;
         for ( int j = cur + 1 ; j < arrayToSort.length ; j++ ) {
            if (arrayToSort[j].compareTo(arrayToSort[min]) < 0) {
               min = j;
            }
         }

         if (min != cur) {
            Integer temp = arrayToSort[min];
            arrayToSort[min] = arrayToSort[cur];
            arrayToSort[cur] = temp;
         }
      }
   }
}
```