Section 1. MCQ questions (4 marks each)

1. Which of the following statements is TRUE?

   A. An interface may implement another interface.
   B. An interface may extend another interface. (correct)
   C. An interface may have concrete implementations for its methods.
   D. Using an interface is the only way to specify an ADT.

2. Consider the following program:

```
class Person {
  public int age;
  public Person(int a) { this.age = a; }
  public void printAge() { System.out.println(this.age); }
}

class Test {
  public static void happyBirthday1(Person p) { p.age++; }
  public static void happyBirthday2(int age) { age++; }
  public static void main(String[] args) {
    Person p1 = new Person(18);
    Person p2 = new Person(21);

    happyBirthday1(p1);
    happyBirthday2(p2.age);
    p1.printAge();
    p2.printAge();
  }
}
```

What is the output?

   A. 18
      21
   B. 19
      22
   C. 18
      22
   D. 19 (correct)
      21
   E. None of the above

3. What will the method call `mystery(958)` return, if the method is defined as follows:

A. 0
B. 1
C. 9
D. 859 (correct)
E. 958

```
public int mystery(int n) {
  int m = 0;
  while (n > 0) {
    m = 10*m + n%10;
    n = n/10;
  }
  return m;
}
```

4. Consider the following code:

```
public class A {
  private int myVar = 0;

  private int GetDoubleVar() { return myVar * 2; }
  int GetVar() { return myVar; }
  protected int GetmyVar() { return myVar; }

  public static void main(String[] args) {
    B b = new B();
    System.out.println(b.GetDoubleParentVar());
  }
}
public class B extends A {
  int myVar = 10;

  // Return 2 times of myVar in the parent class
  public int GetDoubleParentVar() { return xxxxxx; }
  protected int GetmyVar() { return myVar; }
}
```

What are valid replacements for **xxxxxx** that will allow the `GetDoubleParentVar()` method to work correctly?

i) `super.GetDoubleVar()`
ii) `2 * myVar`
iii) `2 * GetmyVar()`
iv) `2 * super.GetmyVar()`

A. i, ii, iii, iv
B. ii, iii, iv
C. iii, iv
D. iv only (correct)
E. none of the above

5. Consider a simple drawing application that can draw shapes such as rectangles and circles. To represent these shapes within the program, you could define a class hierarchy such as this:

```
public abstract class Shape {
  public abstract void draw(Canvas c);
}
public class Circle extends Shape {
  private int x, y, radius;
  public void draw(Canvas c) { ... }
}
public class Rectangle extends Shape {
  private int x, y, width, height;
  public void draw(Canvas c) { ... }
}
```

These classes can be drawn on a canvas:

```
public class Canvas {
  public void draw(Shape s) {
    s.draw(this);
  }
}
```

Any drawing will typically contain a number of shapes. Let's assume that they are represented as an instance of the Java class **`LinkedList`**. It would be convenient to have a method in **`Canvas`** that draws them all. What is the most appropriate method signature for a method **`drawAll`** that achieves this?

A. `public void drawAll(LinkedList<Shape> shapes) { … }`
B. `public void drawAll(LinkedList<? extends Shape> shapes) { … }` (correct)
C. `public void drawAll(LinkedList<Object> shapes) { … }`
D. `public void drawAll(LinkedList<? implements Shape> shapes) { … }`
E. `public void drawAll(LinkedList shapes) { … }`

6. Which of the following cannot be used as the underlying data structure for implementing a stack ADT?

A. Array
B. Queue
C. BasicLinkedList
D. None of the above (correct)

7. Match the blanks to the answers.

a) A software object's state is stored in ___.
b) A software object's behavior is exposed through ___.
c) Hiding internal data from the outside world and accessing it through public methods is called data ___.
d) A ____ is a group of methods and variables.
e) A way to organizes related classes and interfaces by functionality is called a ___.

i. class
ii. package
iii. fields
iv. methods.
v. encapsulation

A. a) iii, b) iv, c) v, d) i, e) ii (correct)
B. a) iii, b) i, c) v, d) iv, e) ii
C. a) iii, b) iv, c) v, d) ii, e) i
D. a) iv, b) iii, c) v, d) i, e) ii
E. a) ii, b) iv, c) v, d) i, e) iii

8. To delete a node N from a linear linked list, you will need to _____.

A. Set the pointer next in the node that precedes N to point to the node that follows N (correct)
B. Set the pointer next in the node that precedes N to point to N
C. Set the pointer next in the node that follows N to point to the node that precedes N
D. Set the pointer next in N to point to the node that follows N
E. None of the above statements is true

9. Consider the following sequence of operations on a stack, which one of them would produce an error (e.g., throw an exception)?

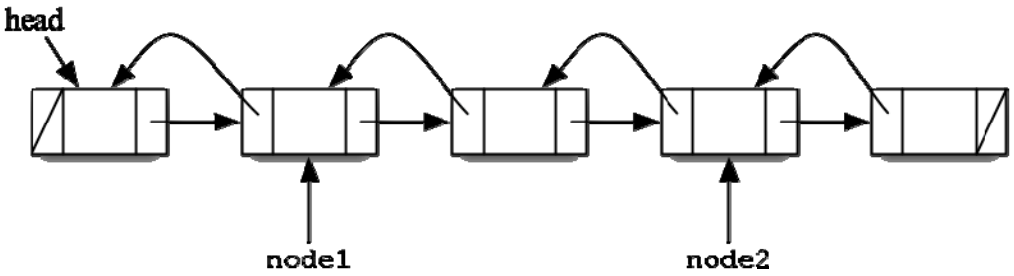
A. create(), push(), push(), pop(), peek(), peek(), peek()
B. create(), push(), peek(), pop(), push(), peek()
C. create(), push(), push(), pop(), peek(), pop(), push(), peek()
D. create(), push(), push(), pop(), peek(), pop(), peek(), push() (correct)
E. create(), push(), peek(), peek(), peek(), pop()

10. Your TA invents a new ADT called StackQueue. A StackQueue consists of three operations: pop (pop out one element from the front), push (add one element to the front), enqueue (append one element to the end). What's more, it is required that the time complexity for all three operations must be efficient. Which of the following data structure you think is the **most suitable** that can be used to implement the StackQueue?

A. Array based list
B. Doubly linked list (correct)
C. Stack (with only pop, push, peek operations).
D. Queue (with only enqueue, dequeue, peek operations).
E. None of the above

Section 2. Short answer / Essay questions (60 total marks)

11. [10 marks] In the doubly linked list below, we would like to switch the list node `node1` with the list node `node2`.



Assume that the next and previous pointers of the nodes can be accessed with `node.next` and `node.prev`. Also, assume that the next and previous pointers of the nodes can be mutated with `node.next = someNode` and `node.prev = someNode`.

Write the code fragment to switch `node1` with `node2`. You may **not** instantiate any temporary list nodes.

Answer:

There are several possible methods. Here is one. Note that the order of operations is important.

```
node1.next = node2.next;
node1.next.prev = node1;
node2.next = node2.prev;
node2.prev.prev = node2;
node2.prev = node1.prev;
node2.prev.next = node2;
node1.prev = node2.next;
node1.prev.next = node1;
```

12. [15 marks] Write a Java method that reads a string and outputs a new string obtained by reversing each word of the original string. Example [Input: "`Life is beautiful`", Output: "`efiL si lufituaeb`"]. The string will not have punctuation nor special characters. You may want to refer to the snippets of the Java API found at the end of the test paper in Appendix A.

Answer:

There are many ways to solve this problem. Here is one verbose method that uses a stack and a queue explicitly.

```
public static String rev (String a) {
    Stack s = new Stack();
    Queue q = new LinkedList(); // LinkedList provides Queue implementation
    String out = "";
    Scanner sc = new Scanner(a);

    while (sc.hasNext()) {  // add tokens to queue
```

```
            q.add(sc.next());
        }

        while (q.peek() != null) {
            Scanner sc2 = new Scanner((String) q.remove());
            sc2.useDelimiter("");
            while (sc2.hasNext()) {
                s.push(sc2.next());
            }
            while (s.empty() == false) {
                out += s.pop();
            }
            out += " ";
        }
        return (out.substring(0,out.length()-1)); // trim final space
    }
```

13. [10 marks] Explain (do not write Java code) how to implement a stack with two queues, by giving explanations for each of **`push`**, **`pop`** and **`peek`** methods. Assuming that **`pop`** and **`peek`** are two critical methods, they should be made as efficient as possible in your design.

Answer:

One of the queues will be used to store the elements (storage queue) and the other (temp queue) to hold them temporarily.

Push: Transfer all the elements in the storage queue to the temp queue, by dequeuing them from the storage and enqueuing them into the temp queue. Then enqueue the given element onto the storage queue. Finally transfer all the elements from the temp queue back to the storage queue.

Pop: return the top element dequeued from the storage queue.

Peek: return the top element of the storage queue by calling getFront.

14. [10 marks] How would you change your solution to question 13 if **`push`** is most critical and must be made as efficient as possible? Again, explain (do not write Java code).

Answer:

Push: Enqueue the given element onto the storage queue.

Pop: Transfer all but the last element from the storage queue to the temp queue, again by dequeuing the elements from the storage and enqueuing into the temp queue. Dequeue the last element from the storage queue and return it. Then restore the storage queue, by transfering all the elements from the temp queue back to the storage queue.

Peek: Similar to Pop. Transfer all but the last element from the storage queue onto the temporary queue, save the front element of the storage queue to be returned, transfer the last element to the temporary queue, and then transfer all elements back to the storage queue.

15. [15 marks] Design a Java method to count the distinct nodes inside an Integer linked list. Two nodes are similar if their elements share the same value; otherwise they are different. You can

assume that the class **`ListNode<E>`** and class **`BasicLinkedList<E>`** are available for you. (These classes' data members and methods are listed at the end of the test paper in Appendix B).

Answer:

There are many possible ways to answer this question. One method shown below is to keep an array of elements that marks whether items on the list have previously been seen. This method does not destroy the original argument to the method.

```
public static int countDistinct(BasicLinkedList<Integer> myLinkList) {
  int count = 0; // Store the final result

  // base case: check if an empty list was given
  if (myLinkList == null) { return 0; }

  // tempArr[i] = 0 if the value of node at index i has not appeared on
  //   any nodes with index smaller than i in myLinkList
  // tempArr[i] = 1, otherwise
  int[] tempArr = new int[myLinkList.size()];
  for (int i=0; i < myLinkList.size(); i++) {
    tempArr[i] = 0; // initialize all entries to 0
  }

  ListNode<Integer> curr = myLinkList.getHead();
  for (int i=0; i < myLinkList.size(); i++) {
    if (tempArr[i] == 1) curr = curr.getNext();
    // Do not need to check again if the node's value has been seen
    else {
      count++; // else increase the counter
      ListNode<Integer> tempPt = curr.getNext();
      // and check for any similar nodes behind this node
      for (int j=i+1; j < myLinkList.size(); j++) {
        if (tempArr[i] == 0 && curr.getElement().equals(tempPt.getElement())) {
          tempArr[j] = 1;
        }
        tempPt = tempPt.getNext();
      } // end for
    } // end else
  } // end for

  return count;
}
```