

1. Suppose we have a class **A** which has a constructor that takes a single integer.

- (a) After the following statements have been executed, how many **A** objects will exist (not counting garbage objects) and which objects are they? Explain your answer and include in your explanation a picture of Java's memory.

```
A a = new A(100);
A b = new A(150);
A c = b;
b = a;
a = null;
```

- (b) After the following statements have been executed, how many **A** objects will exist (not counting garbage objects) and which objects are they? Explain your answer and include in your explanation a picture of Java's memory.

```
A a1 = new A(200);
A a2 = new A(250);
A a3 = a2;
a2 = a1;
a1 = null;
```

2. Here is a simple **Point** and **Circle** class.

```
class Point
{ private double x, y;

  public Point(double x, double y)
  { this.x = x;
    this.y = y;
  }
  public double getX(){ return x; }
  public double getY(){ return y; }
}
```

```
class Circle
{ private Point c; // center
  private double r; // radius

  public Circle(Point c, double r)
  { this.r = r;
    this.c = c;
  }
  // more stuff
}
```

- (a) The constructor in **Circle** has a “privacy leak”. Explain why.

Hint: Consider the following code.

```
Point p = new Point(1,2);
Circle c = new Circle(p, 10);
p.setX(100);
```

- (b) Rewrite the **Circle** constructor to fix this problem.

3. Consider this code that creates some `Location` objects:

```
Location a, b, c;  
a = new Location(10,20);  
b = new Location(10,20);  
c = b;
```

After this code executes, what are the values of these boolean expressions?

```
a==b  
a.equals(b)  
a==c  
a.equals(c)  
b==c  
b.equals(c)
```

Also, write two clear sentences that explain the difference between `==` and the `equals()` method.

4. Consider this code that creates some `Location` objects:

```
Location a, b, c;  
a = new Location(10,20);  
b = (Location)a.clone( );  
c = a;  
c.shift(2,0);
```

After this code executes, what are the values of these boolean expressions?

```
a==b  
a.equals(b)  
a==c  
a.equals(c)  
b==c  
b.equals(c)
```

5. What does the following program print out. Explain why.

```
class Thing
{
    public int a;
    public int b;
    public Thing(int a, int b){this.a=a; this.b=b}
}

public class Test
{
    public static void f(Thing x, int y)
    {
        x.a++;
        y++;
    }
    public static void main(String[] args)
    {
        Thing x = new Thing(1,1);
        int y = 1;
        f(x, y);
        System.out.println("x.a = " + x.a + " and x.b = " + x.b);
        System.out.println("  y = " + y);
    }
}
```

6. Suppose that we have classes A, B, C and D. Suppose that B is a subclass of A, that C is a subclass of B, and D is a subclass of A. Suppose that we make the following declarations.

```
A a1 = new A();
A a2 = new C();
D d1 = new D();
```

For each part below, explain what, if any, errors would be caused by the statement in that part. Be sure to consider both compile time and run time errors.

- (a) A a3 = new B();
- (b) B b1 = new A();
- (c) B b2 = (B) a1;
- (d) B b3 = (B) a2;
- (e) B b4 = (B) d1;
- (f) B b5 = (C)(A)new D();

7. Consider the following classes:

```
public class Organization extends Object {
    Organization() { /* null constructor */ }
    public void printMe() { System.out.println("Organize."); }
}
public class Company extends Organization {
    Company() { /* null constructor */ }
    public void printMe() { System.out.println("Be productive."); }
}
public class MidCap extends Company {
    MidCap() { /* null constructor */ }
    public void printMe() { System.out.println("Think big."); }
}
public class InternetCo extends MidCap {
    InternetCo() { /* null constructor */ }
    public void printMe() { System.out.println("Be cool."); }
    public static void main(String[] args) {
        Company mid = new MidCap();
        MidCap netscape = new InternetCo();
        Object obj = new Organization();
        Organization startup = new Company();
        mid.printMe();
        netscape.printMe();
        ((Organization) obj).printMe();
        obj = netscape;
        ((MidCap) obj).printMe();
        obj = startup;
        ((Organization) obj).printMe();
    }
}
```

What is the output from running the InternetCo class?

8. Suppose we implement the `IntArrayBag` class using two partially-filled, “parallel arrays” instead of a single (partially-filled) array. The first array, `data`, holds the values of the items in the bag and the second array, `dataCounts`, holds a count of the number of times that the associated item is in the bag. In other words, `data[i]` is an integer in the bag, and `dataCounts[i]` is the number of times that integer is in the bag.

We assume that there is an instance variable `manyDataItems` that tells us how many of the entries from the partially-filled arrays `data` and `dataCounts` are used to hold items from the bag (so `manyDataItems <= data.length`). The instance variable `manyItems` is a count of how many items are in the bag.

We assume that the part of the array `data` that stores the bag does not have any duplicate entries and we assume that each value in `dataCounts` is strictly greater than zero for all elements with index less than `manyDataItems`.

- (a) Describe what is meant by the “capacity” of a bag in this implementation.
- (b) Describe an advantage that this implementation of `IntArrayBag` has over the single array implementation from the textbook.
- (c) Describe a disadvantage that this implementation of `IntArrayBag` has when compared to the single array implementation from the textbook.
- (d) Write an implementation for each of the `add(int element)` and `remove(int target)` methods (see the next page).

```
public class IntArrayBag
{ // use two partially-filled "parallel arrays"
    private int[ ] data;           // the data items
    private int[ ] dataCounts;     // how many times each item is in the bag
    private int manyDataItems;     // number of elements in partially-filled array
    private int manyItems;         // total number of items in the bag

    public IntArrayBag( )
    { final int INITIAL_CAPACITY = 10;
      data      = new int[INITIAL_CAPACITY];
      dataCounts = new int[INITIAL_CAPACITY];
      manyDataItems = 0;
      manyItems = 0;
    }

    public IntArrayBag(int initialCapacity)
    { if (initialCapacity < 0) throw new IllegalArgumentException("Capacity<0.");
      data      = new int[initialCapacity];
      dataCounts = new int[initialCapacity];
      manyDataItems = 0;
      manyItems = 0;
    }
}
```

```
/** Add a new element to this bag. If the new element would take this
 * bag beyond its current capacity, then the capacity is increased. */
public void add(int element)
{
```

```
}//add()
```

```
/** Remove one copy of a specified element from this bag.
 * If target was found in the bag, then one copy of target
 * has been removed and the method returns true. Otherwise
 * the bag remains unchanged and the method returns false. */
public boolean remove(int target)
{
```

```
}//remove()
}//IntArrayBag
```

9. Here is part of the definition for a `LinkedList` class.

```
class LinkedList
{
    private ListNode head;
    private int size;

    public LinkedList()
    {
        this.head = null;
        this.size = 0;
    }

    // a private class
    class ListNode
    {
        public int item;          // An item in the list.
        public ListNode next;    // Reference to next item in the list.
    }

    // LinkedList methods...
}
```

(a) Write a method

```
    public void add( int element )
```

that adds a new node at the head of the linked list. (Notice that the inner class `ListNode` only has a default constructor.)

(b) Write a method

```
    public int remove( )
```

that removes from the linked list the node at the head of the list and returns the `int` that was stored in that node. Throw an exception if the linked list is empty.

(c) Explain how you would modify the `add` method so that the following two lines of code will compile.

```
LinkedList list = new LinkedList();
list.add(3).add(2).add(5).add(0).add(8);
```

10. On the last page of these review problems is an implementation of the `IntNode` class.

(a) Write an implementation of the static method

```
public static int countZeros( IntNode node )
```

that will count the number of zeros that occur in the given linked list of ints.

(b) Write an implementation of a static method

```
public static String list2String( IntNode node )
```

that returns a `String` representation of the linked list referred to by the parameter `node`. If the linked list is empty, the `String` representation should be `[]` (two square brackets next to each other). If the linked list is not empty, the `String` representation should look like this, `[3 52 0 2 -4 16]`, with a space before each entry of the list and a space before the closing bracket.

(c) Write a method

```
public static IntNode removeFirst( IntNode head )
```

that returns a reference to the second node from the linked list referred to by the parameter `head`.

(d) Write a method

```
public static IntNode addFirst( int element, IntNode head )
```

that returns a reference to the new head of a linked list with a node containing `element` followed by the list referred to by the parameter `head`.

(e) Write a method

```
public static void set( int element, int i )
```

that modifies the list referred to by the parameter `head` so that the `i`'th node in the list has its data changed to `element`. If there is no `i`'th node in the list, then the list is not modified.

11. Once again using the `IntNode` class, consider the following three lines of code.

```
IntNode head = new IntNode(4,new IntNode(7,new IntNode(5,new IntNode(3,null))));
IntNode ptr = head.getLink().getLink();
head.getLink().setLink( new IntNode(22, null) );
```

- (a) Draw a picture of Java's memory after the first line above has been executed. Be sure to include what data is in each node.
 - (b) Draw a picture of Java's memory after the first and second lines above have been executed.
 - (c) Draw a picture of Java's memory after all three lines above have been executed.
 - (d) What would be a `String` representation for the linked list referred to by `head`?
 - (e) What would be a `String` representation for the linked list referred to by `ptr`?
 - (f) What would be a `String` representation for the linked list referred to by `ptr` after executing the following line (which would be executed after the above three lines)?
`ptr.getLink().setLink(head.getLink());`
12. (a) In the class name `IntArrayBag`, explain the significance of each part of the name: `int`, `array`, and `bag`.
- (b) In the class name `IntArraySeq`, explain the significance of each part of the name: `int`, `array`, and `seq`.
- (c) In the class name `DoubleLinkedBag`, explain the significance of each part of the name: `double`, `linked`, and `bag`.
13. Suppose that

`x = y;`

is a “widening” assignment.

- (a) If `x` and `y` are primitive variables, explain why the assignment can also be referred to as a “widening conversion”.
- (b) If `x` and `y` are reference variables, how are the types of `x` and `y` related to each other?
- (c) If `x` and `y` are reference variables, explain why the assignment should not be referred to as a “conversion”.
- (d) If `x` and `y` are reference variables, you can even make a case that the assignment should be called a “narrowing” assignment. Explain why. (Hint: What can you say about the methods callable on `x` as compared to the methods callable on `y`?)

14. Let A be an array of size $n \geq 2$ containing integers from 1 to $n - 1$, inclusive, with exactly one number repeated.
- (a) Write a method
- ```
public static int findRepeatedNumber(int[] A)
```
- that returns the value of the repeated number in the array  $A$ .
- (b) Rewrite the method so that it uses just a single loop. (Hint: Make use of another array.)
15. Suppose that a **Sequence** ADT has the following interface.

```
public interface Sequence
{ public int size(); // Return number of elements in sequence.
 public void addFirst(int e); // Insert e at the front of the sequence.
 public void addLast(int e); // Insert e at the back of the sequence.
 // Inserts an element e to be at index i.
 public void add(int i, int e) throws IndexOutOfBoundsException;
 // Returns the element at index i, without removing it.
 public int get(int i) throws IndexOutOfBoundsException;
 // Removes and returns the element at index i.
 public int remove(int i) throws IndexOutOfBoundsException;
}
```

Starting with an empty sequence  $A$ , below each operation write down what the (cumulative) contents of the list would be after performing the operation. (Write the contents of the sequence as a horizontal, comma separated, list of numbers with the index 0 element on the left.)

$A.add(0, 4)$

$A.add(0, 3)$

$A.addFirst(2)$

$A.addLast(7)$

$A.add(2, 1)$

$A.add(1, 4)$

$A.add(1, 5)$

$A.add(3, 2)$

16. Below is an outline of a class that implements a linked list of integer nodes with two sentinel nodes (see pages 240–241 of the textbook).

- (a) Draw a picture of the empty list created by the default constructor.
- (b) Write an implementation for the `addFirst()` method.
- (c) Write an implementation for the `removeFirst()` method that assumes the list is not empty.

```
public class IntLinkedList
{ IntNode head;
 IntNode tail;
 int manyItems;

 public IntLinkedList() // Create an empty list,
 { tail = new IntNode(0, null); // with two sentinel nodes.
 head = new IntNode(0, tail);
 manyItems = 0;
 }

 /** Add a new node to the beginning of the list. */
 public void addFirst(int n)
 {

 }

 /** Remove the first node from a nonempty list and return its data. */
 public int removeFirst()
 {

 }

}
} // IntLinkedList
```

```
class IntNode
{
 private int data;
 private IntNode link;

 public IntNode(int data, IntNode link)
 {
 this.data = data;
 this.link = link;
 }

 public int getData() { return data; }
 public IntNode getLink() { return link; }
 public void setData(int data) { this.data = data; }
 public void setLink(IntNode link) { this.link = link; }
} //IntNode
```