

**Problem 1.** (4 points) Quickies

- a. (1 point) Why is it usually better to declare an instance variable **protected** than to declare it **private**?
- b. (1 point) Are **Circle** objects (defined in full below) immutable? Why?

```
public class Circle{
    private double radius;

    public Circle(double r){
        radius = r;
    }

    public double area(){
        final double PI = 3.1415926535897;
        return (2.0 * PI * radius * radius);
    }
}
```

- c. (1 point) In a static method, you can never use the keyword \_\_\_\_\_ .
- d. (1 point) In the following list, circle the types of statements that Java's **continue** statement can potentially jump back to the beginning of.

**for**      **switch**      **if**      **while**      **class**      **do**      **case**

**Problem 2.** (8 points) Inheritance.

```
public class A{
    public A(){
        System.out.println("New A");
    }

    public A(int x) {}

    public void method1() {
        System.out.println("M1 in A");
    }

    public void method2(){
        System.out.println("M2 in " +
            whichClass());
    }

    public String whichClass() {
        return "A";
    }
}

public interface B {
    public void method2();
}
```

```
public class C extends A implements B{
    public C(){
        System.out.println("New C");
    }

    public C(int x){
        super(x);
    }

    public void method1() {
        System.out.println("M1 in C");
    }

    public void method3() {
        super.method1();
    }

    public String whichClass() {
        return "C";
    }
}
```

What does each of the following code fragments print? If a code fragment causes an error, say whether it is a compile-time error or a run-time error. If a code fragment prints nothing and causes no error, say so.

a. `A a = new C(3);`  
`a.method1();`

---

b. `B b = new C(5);`  
`b.method2();`

---

c. `C c = (C) new A(4);`  
`c.method1();`

---

d. `B b = new C(8);`  
`System.out.println(b.whichClass());`

---

e. `B b = new C();`  
`( (A) b ).method1();`

---

f. `(new C(5) ).method3();`

---

g. `B b = new B();`  
`b.method2();`

---

**Problem 3:** (6 points) Binary search.

Recall our binary search code from Lecture 9.

```
public static final int FAILURE = -1;

private static int bsearch(int[] i, int left, int right, int findMe) {
    if(left > right){
        return FAILURE;                //base case 2: subarray of size zero
    }
    int mid = (left + right) / 2;      //halfway between left and right
    if (findMe == i[mid]) {
        return mid;                    //base case 1: success!
    } else if (findMe < i[mid]) {
        return bsearch(i, left, mid - 1, findMe);    //search left half
    } else {
        return bsearch(i, mid + 1, right, findMe);  //search right half
    }
}

public static void main (String[] args) {
    int[] i = {0, 3, 7, 9};
    int j = bsearch(i, 0, i.length - 1, 7);
}
```

Suppose we execute **main** above. Draw the stack and heap at the moment when the topmost **bsearch** call on the stack is about to return **2**. Specifically, draw a box-and-pointer diagram with a box for every stack frame, local variable, object, and field in memory at that moment. Include the stack frames for all methods in progress, and illustrate which entities are inside those stack frames. Entities on the stack should be on the left-hand side of the page, and entities on the heap should be on the right-hand side.



Draw stuff on the stack on the left side

Draw stuff on the heap on the right side

**Problem 4.** (7 points) Reordering a Singly-Linked List.

Write a method called **reorder** in the **SList** class below. **reorder** changes the order of nodes in a singly-linked list. Upon completion, the nodes in the list appear in the following order: the former first node, the former third node, the former fifth node, followed by the remaining former odd-numbered nodes. Then comes the former second node, the former fourth node, and so on.

For example, [ **This is a list of coherent strings** ]  
becomes [ **This a of Strings is list coherent** ].

Your solution should manipulate **next** pointers directly. Do not call any method of **SList** provided in this course. Do **not** change any **item** references or create any new nodes.

```
public class SListNode{
    public Object item;
    public SListNode next;
}

public class SList{
    private SListNode head;
    private int size;

    public void reorder(){

    }
}
```