Computer Science 241
Test 2
Apr 11, 2005

1. Indicate for each method invocation on the right which method is called from the classes on the left. (2 points each)

```
public class A {
    public int m(A a) { ... }  // #1

    public int m(B b) { ... }  // #2

    public int m(C c) { ... }  // #3

    public int m(D d) { ... }  // #4
}


public class B extends A {
    public int m(A a) { ... }  // #5

    public int m(B b) { ... }  // #6

    public int m(C c) { ... }  // #7
}


public class C extends B {
    public int m(A a) { ... }  // #8

    public int m(B b) { ... }  // #9
}


public class D extends C {
    public int m(A a) { ... }  // #10
}
```

```
A a = new B();
B b = new C();
C c = new D();
```

`a.m(c);`	7
`((B) a).m(c);`	7
`b.m(a);`	8
`b.m((B) a);`	9
`c.m((D) c);`	4
`c.m(c);`	7
`c.m(b);`	9
`c.m(a);`	10

}

2. Consider the interface on the back page with its documentation. Fill in the missing parts of the following class so that it fulfills the contract implied in the interface's documentation. (20 points)

```
public class IntArray implements IntContainer {

        private int[] array;

        // Constructor where size is the size of the container
        public IntArray(int size) {


                    array = new int[size];



        }



        public void add(int index, int value) {


                    if (index >= 0 && index < array.length)
                        array[index] = vale;



        }



        public int average() {


                    int sum = 0;
                    for (int i = 0; i < array.length; i++)
                        sum += array[i];
                    return sum / array.length;


        }
}
```

3. Find examples in the following code. (3 points each)

```
( 1)   interface I {
( 2)       public int m1(int x);
( 3)   }
( 4)   class A {
( 5)       protected int z;
( 6)       public A(int z) {
( 7)           this.z = z;
( 8)       }
( 9)       public int m1(int y, I i) {
(10)           return y + i.m1(z);
(11)       }
(12)       public int m2(int y, A a) {
(13)           return y + a.z;
(14)       }
(15)   }
(16)   class B extends A implements I {
(17)       public B(int z) { super(z); }
(18)       public int m1(int x) {
(19)           return x;
(20)       }
(21)       public int m1(int y, I i) {
(22)           return y - i.m1(z);
(23)       }
(24)   }
(25)   class C extends B {
(26)       private A aa;
(27)       public C(int z) {
(28)           super(z);
(29)           aa = new B(z);
(30)       }
(31)       public int m1(int y, A a) {
(32)           if (y < 0)
(33)               aa = a;
(34)           return y + a.z;
(35)       }
(36)   }
```

Give an example of aliasing. `33:  aa and a are aliased`

Give an example of shadowing. `6-8:  parameter z shadows instance variable z`

Give an example of a receiver. `i in line 22`

Give an example of a primitive type. `int in line 2`

Give an example of a reference type. `A`

Give an example of a composite type. `A`

Give an example of when an expression has a static type that differs from its dynamic type. `aa has static type A and dynamic type B`

Give an example of method overloading. `B overloads m1:  18 and 21`

Give an example of method overriding. `21:  B overrides m1 from A`

Give an example of an “is-a” relationship. `B ‘‘is a’’ A`

4. Look at the following code and find four errors. (4 points each)

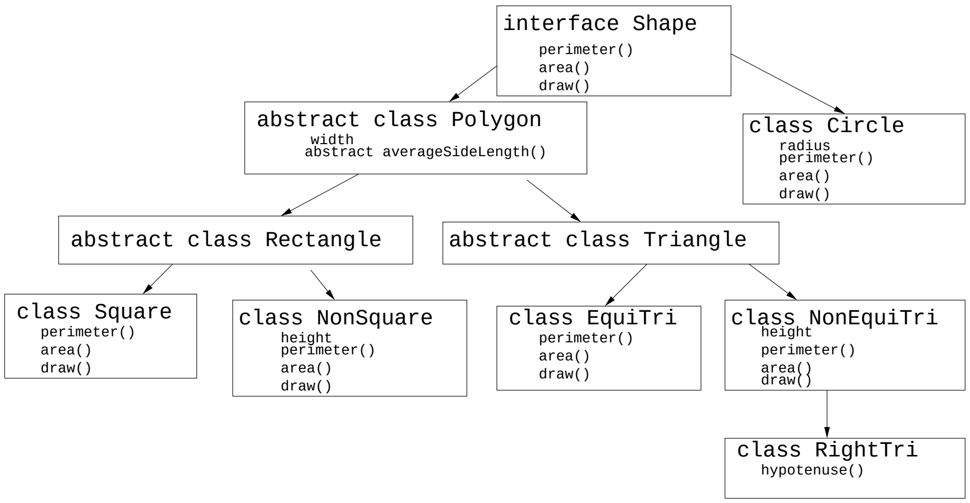
```
public abstract class A {
     private int x;

     public abstract int m1();

     public final String m2(int y) {
          return x + " /  " + y;
     }
}

public class B extends A{
     public String m2(int y) {
          A a = new B();
          return a.x +  " /  " + a.m3(y);
     }

     public int m3(int z) {
          return z + 3;
     }
}
```

```
m2 is final so B cannot override it.
x is private, so B cannot have ‘‘a.x’’.
a has static type A and so ‘‘a.m3(y)’’ is not valid at compile time.
B has no m1, even though it extends A (with abstract method m1) and is not
itself abstract.
```

5. You are writing an application that manipulates shapes. You want to support the following shapes: rectangle (including squares), circles, and triangles (including right and equilateral). Design a class hierarchy; for each type, indicate whether it is an interface, class, or abstract class, what instance variables and methods it has, and which methods are abstract. Fulfill the following constraints:

- For all shapes, you can compute the perimeter (ignore the fact that for circles, the *perimeter* is usually called the *circumference*), compute the area, and draw the shape.
- For all polygons, you can compute the average side length.
- For right triangles, you can compute the hypotenuse.
- [**Extra credit for fulfilling this constraint**] For the sake of efficiency, squares and equilateral triangles should store only one side length (since all sides are the same).

There is more than one correct answer. (18 points)



```
/**
 * IntContainer.java
 *
 * Interface for a class that contains a finite sequence of integers
 * and can compute their average.
 *
 * @author Duane Litfin
 * Wheaton College, CS 241, Spring 2005
 * Test 2
 * April 11, 2005
 */
public interface IntContainer {

     /**
      * If index is in the range of valid indices, set the value of
      * the sequence at that index to be value; otherwise, do nothing.
      * @param index The position in the collection we are setting
      * @param value The value we are setting at the given position
      */
     public void add(int index, int value);

     /**
      * Calculate the average of all values in the sequence; if there
      * are positions that have not been set, they are assumed to be zero
      * and they are still included in the average.
      * @return The average, as an integer.
      */
     public int average()
}
```