

Name:

Account:

Practice Exam

1. Java dynamic dispatch (20 points)

Given the following Java class definitions:

```
public class A {
    A() { System.out.println("A.A()"); }
    public void f(A) { System.out.println("A.f(A)"); }
    public void f(B) { System.out.println("A.f(B)"); }
    public void g(A) { System.out.println("A.g(A)"); }
};

public class B extends A {
    B() { System.out.println("B.B()"); }
    public void f(A) { System.out.println("B.f(A)"); }
    public void f(B) { System.out.println("B.f(B)"); }
    public void g(B) { System.out.println("B.g(B)"); }
    public static void h(A) { System.out.println("B.h(A)"); }
}
```

What are the effects of each of the following statements executed in sequence (e.g., prints xxx, causes a run-time exception to be thrown, gives a compile-time error, etc.)?

- `A a = new A();`
prints A.A()
- `B b = new B();`
prints A.A()
prints B.B()
- `A ab = new B();`
prints A.A()
prints B.B()
- `a.f(ab);`
prints A.f(A)
- `a.g(b);`
prints A.g(A)
- `b.f(ab);`
prints B.f(A)

- `b.g(a);`
prints `A.g(A)`
- `b.h(ab);`
prints `B.h(A)`
- `ab.f(a);`
prints `B.f(A)`
- `ab.h(a);`
compile-time error - no function `A.h(A)`
- `ab.f(ab);`
prints `B.f(A)`

Name:

Account:

2. Java exceptions (20 points)

Given the following code:

```
class L1Exception extends Exception {
    public L1Exception() {}
    public L1Exception(String msg) { super(msg); }
}

class L2Exception extends L1Exception {
    public L2Exception() {}
    public L2Exception(String msg) { super(msg); }
}

public class ExceptionTest{
    public static void f(int x) throws L1Exception,L2Exception {
        if (x > 0) { throw new L1Exception(); }
        else { throw new L2Exception(); }
    }

    public static void g(int y) throws L2Exception,L1Exception {
        if (y <= 0) { throw new L1Exception(); }
        else { throw new L2Exception(); }
    }

    public static void main(String[] args) throws L1Exception, L2Exception {
        int a = Integer.parseInt(args[0]), b=Integer.parseInt(args[1]);
        try { f(a); g(b); }
        catch (L1Exception e) { System.out.println(e); }

        try { g(b); }
        catch(L2Exception e) { System.out.println(e); }
        finally {
            if (a <= 0) f(a);
            System.out.println("quitting");
        }
    }
}
```

Name:

Account:

What is the effect of executing ExceptionTest with the following arguments (e.g., prints xxx, throws a run-time exception of type E, etc.)?

- 1 1
prints L1Exception
prints L2Exception
prints "quitting"

- 1 -1
prints L1Exception
prints quitting
throws L1Exception

- -1 1
prints L2Exception
prints L2Exception
throws L2Exception

- -1 -1
prints L2Exception
throws L2Exception

Name:

Account:

3. Java multithreading (20 points)

In the code given below the `ProducerConsumerTest` class creates two threads: one of class `Producer` and another of class `Consumer`. The consumer thread queries the user for a data item and a buffer id. It then stores this data in the given buffer. The `Consumer` thread queries a user for a buffer id. Upon receiving it, the `Consumer` thread requests the data associated with that id.

In order to this both threads use a shared instance of the `CubbyHole` class. In particular they call `CubbyHole`'s `put` and `get` functions.

Your task is to write `CubbyHole`'s `put` and `get` methods. To work properly `CubbyHole` must ensure that no data is lost or duplicated. That is, once a data item is written to a buffer (with `put`) it must remain unchanged until it is consumed (with `get`). The data item can be consumed exactly once.

Every call to `put` must succeed. That is, `put` returns only after the data item has been properly stored. On the other hand, `get` will return a value of -1 when a buffer id has no data or when the storage associated with the buffer id is being written to.

I have included nearly all the code for this application. `Producer`, `Consumer`, and `ProducerConsumerTest` do obvious things. You shouldn't need to spend more than a minute looking at them. Focus on `CubbyHole`.

NOTE: You are **not** allowed to change any of the code that's written below.

```
class CubbyHole {

    class Semaphore {
        boolean filled;
        Semaphore (boolean b) {
            filled = b;
        }
    }

    private int seq[];
    private Semaphore available[];
    private int numBins;

    CubbyHole(int i) {
        numBins = i;
        seq = new int [numBins];
        available = new Semaphore [numBins];
        for (int j = 0; j < i; j++)
            available[j] = new Semaphore (false);
    }

    public int get(int bin) {
        // Write the code that goes here
    }

    public void put(int value,int bin) {
        // Write the code that goes here
    }
}
```

```

class Producer extends Thread {
    private CubbyHole cubbyhole;

    public Producer(CubbyHole c) {
        cubbyhole = c;
    }

    public void run() {
        int i,j;
        while (true) {
            i = getValue(); // gets value from user
            j = chooseBin(); // gets buffer ID from user (0..numBins-1)
            cubbyhole.put(i,j);
            System.out.println(i + " " + j);
        }
    }
}

class Consumer extends Thread {
    private CubbyHole cubbyhole;

    public Consumer(CubbyHole c) {
        cubbyhole = c;
    }

    public void run() {
        int value, bin;
        while (true) {
            bin = chooseBin(); // gets buffer ID from user (range 0..numBins-1)
            value = cubbyhole.get(bin);
            System.out.println(bin + " " + value);
        }
    }
}

class ProducerConsumerTest {
    public static int numBins = 10;
    public static void main(String args[]) {
        CubbyHole c = new CubbyHole(numBins);
        Producer p1 = new Producer(c);
        Consumer c1 = new Consumer(c);
        p1.start();
        c1.start();
    }
}

```

```
public int get(int bin) {
    int value;
    synchronized(available[bin]) {
        if (!available[bin].filled) {
            value = -1;
        }
    }
    else {
        available[bin].filled = false;
        (available[bin]).notify();
        value = seq[bin];
    }
}
return value;
}

public void put(int value,int bin) {
    synchronized(available[bin]) {
        while (available[bin].filled) {
            try { (available[bin]).wait(); }
            catch (InterruptedException e) { }
        }
        seq[bin] = value;
        available[bin].filled = true;
        (available[bin]).notify();
    }
}
```

Name:

Account:

4. (Threads) 20 points

What are the possible outputs of the following Java code, if it is executed by invoking **java PrintTest** from the Unix shell?

```
class Print {
    String str;
    Print(String s) {
        str = s;
    }
    public void run(){
        System.out.println(str);
    }
}

public class PrintTest {
    public static main(String[] args) {
        Thread t1 = new Thread(new Print("Thread 1"));
        Thread t2 = new Thread(new Print("Thread 2"));
        Thread t3 = new Thread(new Print("Thread 3"));

        t3.start();
        t2.start();
        t1.start();
    }
}
```

6 possible outputs. All combinations of the three strings.