Honor Pledge

On my honor, I have neither given nor received any unauthorized aid on this test.

Specify the exact time (including time zone) when you first accessed the contents of this test in the box below.

By typing your first and last name in the space provided below you are electronically signing to indicate that:

1. You are the person who is taking this test.
2. You read and understood the Honor Pledge and you agree to be bound by it.
3. You will submit your responses no later than two hours from the time you first accessed the contents of this test, unless you are entitled to receive extra time accommodations (in which case you will submit your responses in accordance with the accommodations you are receiving).

If your first and last name does not appear below, your test will not be graded and you will receive a grade of zero.

Question 1. (15 pts) Willy Wazoo is implementing an immutable polynomial with integer coefficients using an IntMap. All specifications are correct; Willy's code, not necessarily so.

```
import java.util.Map;
public class Poly {
    // E.g., { <2,1>,<1,-2>,<0,1> } represents polynomial x^2 - 2*x + 1.
    // An empty map represents the ZERO polynomial.
    private Map<Integer,Integer> expToCoeff; // rep: exponent-to-coefficient map

    // requires: input is such that it satisfies rep invariant of Poly
    // effects: creates a new Poly represented by input map
    public Poly(Map<Integer,Integer> input) {
        expToCoeff = input;
  }

  // requires: q != null
    // returns: a new Poly r such that r = this + q
    public Poly add(Poly q) {
    for (Integer key : q.expToCoeff.keySet()) {
      int c1 = 0;
      int c2 = q.expToCoeff.get(key);
      if (expToCoeff.containsKey(key))
        c1 = expToCoeff.get(key);
      expToCoeff.put(key,c1+c2);
       }
    return new Poly(expToCoeff);
    }
}
```

Write a suitable representation invariant below:

Write a suitable abstraction function below:

List everything that is wrong with Willy's code. (Since you wrote the rep invariant and abstraction function, not Willy, assume they are correct, and that the code should be written against them.)

Question 2. (10 pts, 2 pts each) Choose the better one: checked or unchecked exception?
LinkedList.set when the index is negative

File.open when there is a disk failure

Integer.parseInt when the argument doesn't represent a number

ArrayList.ensureCapacity when the system is out of memory

File.open when the file does not exist

Question 3. (10 pts, 2 pts each) Select **true** or **false**.
It is acceptable practice to use statements that have side effects in assert conditions (e.g., assert set.add(date);)

A RuntimeException typically indicates an unrecoverable programming error

Unchecked exceptions cannot be caught using the standard try catch finally syntax

Only objects of type Throwable or subclasses of Throwable can be used as exceptions in Java

It is better practice to check parameters of public methods with preconditions and assertions, rather than throw exceptions

Question 4. (15 pts, 3 pts each) For each of the following code snippets below, select one of the options:

- **"True Function Subtype"**, if the method in the subclass is a true function subtype
- **"OK"**, if the subclass method is valid Java code, but not a true function subtype
- **"Error"**, if the code would result in a compiler error

```
class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Integer m(Number x, Number y) { return super.m(x, y); }
}
```

```
class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Integer m(Number x, Number y) { return null; }
}
```

```
class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Object m(Number x, Number y) { return null; }
}
```

```
class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Number m(Integer x, Integer y) { return y; }
}
```

```
class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Object m(Number x, Number y, Number z) { return y; }
}
```

Question 5. (18 pts, 3 pts each) Consider the code below:

```
class A {
  void f() { System.out.println("A.f()"); }
  void f(int n) { System.out.println("A.f(int)"); }
  void g(int n) { System.out.println("A.g(int)"); }
}
class B extends A {
  void f() { System.out.println("B.f()"); }
  void h() { System.out.println("B.h()"); }
}
class C extends B {
  void g(int n) { System.out.println("C.g(int)"); }
  void g(String s) { System.out.println("C.g(String)"); }
  void h(int n) { System.out.println("C.h(int)"); }
}
```

For each of the lines of code below, show what is printed or indicate that the the line cannot be executed because of a compiler error by typing **"Error"**. Each line of code with the pair of statements is independent of the others.

```
A a = new C(); a.f();
```

```
A a = new C(); a.g(1);
```

```
A a = new C(); a.g("cat");
```

```
B b = new B(); b.g(1);
```

```
B b = new C(); b.h();
```

```
B b = new C(); C c = b;
```

Question 6. (12 pts, 3 pts each) Suppose we have the following code for a class that represents circles in a 2-D graphics system.

```
public class Circle {
  private int x; // x and y center coordinates
  private int y;
  private int radius; // radius
  // two Circles are considered to be equal if they
  // have the same center coordinates
  public boolean equals(Object other) {
    if (! (other instanceof Circle)) return false;
    Circle c = (Circle) other;
    return this.x == c.x && this.y == c.y;
  }
...
}
```

For each of the following hashCode methods, indicate which meet the requirements for a correct hashCode for `Circle.equals()` by selecting **"OK"** or **"NOT OK"**.

```
public int hashCode() {
  return x;
}
```

```
public int hashCode() {
  return x*x + y*y;
}
```

```
public int hashCode() {
  return x + y + radius;
}
```

```
public int hashCode() {
  return -1;
}
```

Question 7. (15 pts) Consider the following code:

```
import java.math.BigDecimal;
public class Star {

  public Star(String name, BigDecimal radius){
    this.name = name;
    this.radius = radius;
  }

  public final String getName(){ return name; }
  public final BigDecimal getRadius() { return radius; }

  private String name;
  private BigDecimal radius;
  private int hashCode;
}

class Pulsar extends Star {

  public Pulsar(String name, BigDecimal radius, BigDecimal radioPeak){
    super(name, radius);
    peakRadioFreq = radioPeak;
  }

  public final BigDecimal getPeakRadioFrequency(){
    return peakRadioFreq;
  }

  /** Peak radio frequency, in Hertz. */
  private BigDecimal peakRadioFreq;
  private int hashCode;
}
```

Write a meaningful implementation for the `equals()` method which allows comparing `Star` and `Pulsar` objects by value in any combinations. Equality should also work correctly even if one or both variables are `Object`. Whenever possible, adhere to the equality and identity guidelines that we discussed in class.

Select all properties which hold for your implementation of `equals()`:

Now, implement `hashCode()` method so that it is valid and matches your implementation of `equals()`. As an additional requirement, your code should only compute the value of hashcode if it hasn't been computed for this object before (so called lazy initialization). If it has been computed before, `hasCode()` should return this previously computed (cached) value.

Finally, rewrite `Star` and `Pulsar` classes using composition instead of inheritance. Do not forget to implement all methods that `Star` and `Pulsar` had in the original version, as well as `equals()` and `hashCode()`. For this implementation, `equals()` should only allow comparing a `Star` object to another `Star` object and a `Pulsar` object to another `Pulsar` object.

Question 8. (5 pts) Consider the following code:

```
// returns the largest of the 3 arguments
int max3(int x, int y, int z) {
  int a;
  if(x > y) {
    a = x;
  } else {
    a = y;
  }
  if(z > a) {
    return z;
  } else {
    return x;
  }
}
```

Give a minimal test suite for this method with full branch coverage and where all tests in the suite pass.

Give a test that does not pass or write “None” if all possible tests would pass.