Question 1 (15 points)

5 for the correct rep. invariant, 5 for the correct abstraction function, 5 for listing everything that is wrong with Willy's code

Representation invariant:

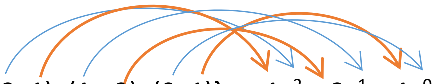
expToCoeff != null && for all k in keys of expToCoeff k >=0 && for K, the set of keys of expToCoeff, for k_{max} such that for all k in keys of expToCoeff k <= k_{max}, for k_{min} such that for all k in keys of expToCoeff k >= k_{min}, | K | == k_{max} - k_{min} + 1 && expToCoeff[k_{max}] != 0 && for all v in values of expToCoeff v != null

Having more parts in the representation invariant than in our solution is OK, so long as they are correct.

Abstraction function:

A key from expToCoeff maps to exponent of the polynomial, and the value maps to the coefficient for the same term:

{(e_m, c_m), .., (e_l, c_l), … (e_k, c_k)} => $c_m\ x^{e_m}$ + … + $c_l\ x^{e_l}$ + … + $c_k\ x^{e_k}$



For example, {(2, 1), (1, -2), (0, 1)} => $1x^2 – 2x^1 + 1x^0$

An empty map represents the ZERO polynomial: {} => 0

There might be different ways to express an abstraction function. Any correct method of defining the abstraction function would be fine (e.g., pictorially wouldn't be an option given the Submitty gradeable).

Issues with Willy's code:

- Rep. exposure in constructor
- add() rep exposure from passing a reference field directly into a new constructor
- add() does not check if the largest term is zero to shrink the size (i.e., might break the "for k_{max} such that for all k in keys of expToCoeff k <= k_{max} expToCoeff[k_{max}] != 0" part of the invariant)
- add() modifies expToCoeff which contradicts the "modifies" spec of add()

There might be other issues, as well.

Question 2. (10 points, 2 points each)

LinkedList.set	checked
File.open when there is a disk failure	unchecked
Integer.parseInt	checked
ArrayList.ensureCapacity	unchecked
File.open when the file does not exist	checked

Question 3. (10 points, 2 points each)

It is acceptable practice to use statements that have side effects in assert conditions (e.g., assert set.add(date);)	false
A RuntimeException typically indicates an unrecoverable programming error	true
Unchecked exceptions cannot be caught using the standard try catch finally syntax	false
Only objects of type Throwable or subclasses of Throwable can be used as exceptions in Java	true
It is better practice to check parameters of public methods with preconditions and assertions, rather than throw exceptions	false

Question 4. (15 points, 3 points each)

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); }
}
	Error (the call to super returns a Number, can't convert from a Number to an Integer)

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

class A {
  Number m(Number x, Number y) { return x; }
}
class B extends A {
  Object m(Number x, Number y) { return null; }
}
	Error
	Looks like it would be an override, but return type is not covariant. If this were OK, it could happen:
	`A a= new A();`
	`B b = new B();`
	`Number n = a.m(x,y);`
	`n = b.m(x,y);`
	The last statement fails because you can't convert an Object to a Number. B.m() is not substitutable for A.m().

class A {
  Number m(Number x, Number y) { return x; }
}
	OK (overload)

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; }
}
	OK (overload)

Question 5. (18 points, 3 points each)

`A a = new C(); a.f();`	B.f()
`A a = new C(); a.g(1);`	C.g(int)
`A a = new C(); a.g("cat");`	Error
`B b = new B(); b.g(1);`	A.g(int)
`B b = new C(); b.h();`	B.h()
`B b = new C(); C c = b;`	Error

Question 6. (12 points, 3 points each)

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

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

`public int hashCode() {`
`  return x + y + radius;`
`}`
	NOT OK (Two circles might have the same x and y coordinates but different radii)

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

Question 7 (15 points)

4 for the correct equals(), 2 for correctly identified properties of equals(), 4 for the correct hashCode(), 5 for the correct implementation using composition.

equals()

Any valid solution which meets the requirements would be accepted:

- Allows comparing Star and Pulsar objects by value in any combinations.
- Equality should also work correctly even if one or both variables are Object.

- The signature is exactly boolean equals(Object).
- Valid code (compiles and runs).

<u>One solution:</u>

Star class

```
public boolean equals(Object o) {
  if (!(o instanceof Star))
    return false;
  Star s = (Star) o;
  return this.name.equals(s.name) && this.radius.equals(s.radius);
}
```

Pulsar class

```
public boolean equals(Object o) {
  if (!(o instanceof Star))
    return false;
  if (!(o instanceof Pulsar))
    return super.equals(o);
  Pulsar p = (Pulsar) o;
  return super.equals(p) && peakRadioFreq.equals(p.peakRadioFreq);
}
```

<u>Another possible solution which implies that name uniquely identifies a celestial object, be that a star or a pulsar:</u>

Star class

```
public final boolean equals(Object o) {
  if (!(o instanceof Star))
    return false;
  Star s = (Star) o;
  return this.name.equals(s.name);
}
```

Pulsar class

No need to override equals()

Properties of equals()

For our solution above the following properties hold.

One solution:

Reflexive	true
Symmetric	true
Transitive	false
Consistent	true

Another possible solution:

Reflexive	true
Symmetric	true
Transitive	true
Consistent	true

hashCode()

Any valid solution which meets the requirements would be accepted. Most importantly, the following requirements should be met for hashCode():

- hashCode() is consistent with the implementation of equals(), i.e., when equals() returns true, hashCode() should return the same value for both objects
- hashCode() computes the value only once on the first invocation; subsequent calls should retrieve this value from cache
- The signature is exactly int hashCode()
- Valid code (compiles and runs)

One solution:

Star class

```java
public int hashCode() {
  if (hashCode == 0) {
    hashCode = 7 * this.name.hashCode() + 11 * this.radius.hashCode();
  }
  return hashCode;
}
```

Pulsar class

No need to override hashCode() but if it is overridden when a Star is equal to a Pulsar in the sense of the equals() method, both objects return the same hashCode().

Another possible solution which implies that name uniquely identifies a celestial object, be that a star or a pulsar:

Star class

```
public final int hashCode() {
  if (hashCode == 0) {
    hashCode = this.name.hashCode();
  }
  return hashCode;
}
```

Pulsar class

No need to override hashCode() but if it is overridden when a Star is equal to a Pulsar in the sense of the equals() method, both objects return the same hashCode(). Also, hashCode() should not take any fields except name into account in order to be consistent with equals().

Star/Pulsar using composition

Any valid solution which meets the requirements would be accepted:

- No inheritance. Star should be enclosed in Pulsar as a field.
- equals() and hashCode() for both classes should be valid (see the requirements above). Pay special attention that equals() and hashCode() are valid given how the Pulsar class was rewritten.

Different correct solutions are possible. Here’s one:

```java
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; }

  public final boolean equals(Object o) {
    if (!(o.getClass().equals(getClass())))
      return false;
    Star s = (Star) o;
    return this.name.equals(s.name) && this.radius.equals(s.radius);
  }

  public final int hashCode() {
    if (hashCode == 0) {
      hashCode = 7 * this.name.hashCode() + 11 * this.radius.hashCode();
    }
    return hashCode;
  }

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

class Pulsar{

  public Pulsar(String name, BigDecimal radius, BigDecimal radioPeak){
    star = new Star(name, radius);
    peakRadioFreq = radioPeak;
  }

  public final String getName() { return star.getName(); }

  public final BigDecimal getRadius() { return star.getRadius(); }

  public final BigDecimal getPeakRadioFrequency(){
    return peakRadioFreq;
  }
  public final boolean equals(Object o) {
    if (!(o.getClass().equals(getClass())))
      return false;
    Pulsar p = (Pulsar) o;
    return star.getName().equals(p.getName()) &&
star.getRadius().equals(p.getRadius())
      && peakRadioFreq.equals(p.peakRadioFreq);
  }

  public final int hashCode() {
    if (hashCode == 0) {
      hashCode = 7 * star.getName().hashCode() + 11 * star.getRadius().hashCode()
+
        13 * peakRadioFreq.hashCode();
    }
    return hashCode;
  }

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

Question 8 (5 points)

3 for the minimal test suite, 2 for the test that doesn’t pass.

Minimal test suite

Different correct solutions are possible. Here’s one:

```
max3(2,3,5);
max3(8,3,5);
```

Test that doesn’t pass

Different correct solutions are possible. Here’s one:

```
max3(2,13,5);
```