Question 1.

```
/** See prelim 2 for the specification */
public static int eqChars(String s, int i) {
    int k=  1;
    /** inv: characters in s[i..i+k-1] are the same. */
    while (i+k < s.length()  &&
              s.charAt(i) == s.charAt(i+k)) {
        k= k+1;
    }
    return k;
  }
```

Question 2.

```
/** See prelim 2 for the specification */
public static String compress(String s) {
    if (s.length() == 0)
       return s;
    // s has at least one character
    int len= eqChars(s, 0);
    return s.charAt(0) +
        len + compress(s.substring(len));
}
```

Question 3. (a)

b.p1(5) no. c.p1(5) yes. d.p1(5) yes.

b.p1() no. c.p1() no d.p1() yes.

(b) (C2 (b)).p2(3, 4)

(c)

```
/** See prelim 2 for the specification */
public boolean equals(Object b) {
    if (!(b instanceof Planet)) {
       return false;
    }
    Planet bp= (Planet) b;
    return super.equals(b)  &&  ms.equals(bp.ms);
}
```

Question 4.

(a) Making a class abstract means that it cannot be instantiated (objects of it cannot be created).

(b) Making a method abstract forces all non-abstract subclasses to override the method.

(c)

```
public abstract class A {
    public abstract void a() {}
  }

public class B extends A {
  public void a() {}
  public void h() {}
}

public class C extends A {
  public void a() {}
}

public class D extends B { }
```

We describe how we drew conclusions from the four points.

1. a= **new** A(); // compile-time error
   a.h(); // compile-time error

A is abstract and does not declare h().

2. These compile and run without error:

```
d= new D();  a= d;  b= (B)a;
a.a();  b.h();  d.h();
```

D extends B and B extends A. A declares a(). B declares h(). D may or may not declare h().

3. **class** E **extends** A { } gives compile-time error.

Happens only if A.a() is abstract, forcing E to declare a(). So A.a() is abstract. Therefore B overrides a().

4. No object has partitions for all of A, B, C, and D.

Therefore, C does not extend D and does not fit anywhere in the hierarchy D -> B -> A -> Object.

5. a= **new** C(); compiles and runs, and c= (B)a; compiles but produces a runtime error.

Therefore C extends A or B; the runtime error indicates C cannot extend B, so C extends A. C overrides a().

The required declared methods are A.a(), B.h(), B.a(), and C.a(); since exactly four methods are required, D does not declare h().

Question 5.

(a) String[] bb= {"a", "e", "i", "o", "u"};

(b)

```
/**See prelim for the specification */
public static Vector<CelestialBody>
                          extract(CelestialBody[] b) {
  Vector<CelestialBody> v= new
                           Vector<CelestialBody>();
  // Store in v a list of elmts of
     b[0..length-1] but with no dups
  // inv: v contains elmts of b[0..k-1] but with no dups
  for (int k= 0; k < b.length; k= k+1) {
       if (k == 0 || b[k] != b[k-1]) {
          v.add(b[k]);
       }
   }
   // v contains elmts of b but with no dups
   return v;
}
```