Shira Pahmer: 575285308

Maeven Fanebust: 559816034

Part I:

A-

public interface A{  
 void f();  
 void g();  
 void h();  
}

B-  
public class B extends C{  
 int x;  
 string y;  
  
 public B(){};  
 public B(int x, string y){};  
}  
  
public class C implements A{  
 public C(){};  
  
 public void f(){};  
 public void g(){};  
 public void h(){};  
}

C- *A*

void f()  
 void g()  
 void h()

C B

Void f() Int x  
 void g() string y  
 void h()

D- We know that A is an interface and has the functions f, g and h in them because the last three lines of the code have the objects of type A calling those functions. We know that B is a type of C because we declare object b1 of type C but set it equal to a new B object. And because we create new B and C objects to put in the array of A objects, we know they must implement A and because B already has to extend C that means C can implement A and B will also get A’s functions from C’s implementation.

Part II D-

Because we implemented our own checks inside the compareTo() functions of Plane and Train, if an object that is not a matching type to the calling object is passed in the function returns 0. The sort will then think they are the same object and not move them.