1. EXTENDING EXISTING CLASSES AND WRITING TESTS [20 points]
Consider the `BooleanCollection` class, which is identical to what you saw in Assignment 3 and appears on the first page of the code section of the exam.

(1.a) [4 points] Write a *specification* for the following method to be added to the `BooleanCollection` class:

> The method `removeThese` takes a single parameter, a boolean `b`, and removes from the collection all those elements that are equal to `b`. So, if `b` is true then `removeThese` removes all the true elements and if `b` is false then `removeThese` removes all the false elements.

```
// MODIFIES: this
// EFFECTS: removes from the collection all values that
// are equal to b
public void removeThese(boolean b) {
    // stub
}
```

KEY:
[2] – modifies & effects clauses (1 mark each)
[2] – parameter list & return type (1 mark each)

(1.b) [6 points] We would like to add a `sameUpTo` operation to our `BooleanCollection` class which returns true if this `BooleanCollection` and one passed as an argument have the same elements in the same order up to but not including the element at the provided `index`. The specification for the `sameUpTo` method is:

```
// REQUIRES: 0 <= index <= smallest of: getNumberOfItems() and
// other.getNumberOfItems()
// EFFECTS: return true if this and other have the same elements in
// the same order, up to but not including the element at the
// provided index, false otherwise
public boolean sameUpTo(BooleanCollection other, int index) {

    for (int i = 0; i < index; i++) {
        if (get(i) != other.get(i))
            return false;
    }

    return true;
}
```

KEY:
[3] – for loop (deduct [1] mark for each error)
[2] – branch logic in body of loop (no part marks)
[1] – final return statement correct

(1.c) [4 points] Write a single test method that tests the `BooleanCollection`'s `sameUpTo` method when both `this` and `other` are empty and the argument `index` has the value 0. This method would be in, for example, a `BooleanCollectionTest` class. Assume that no `@Before` method exists - if you wish to make use of one, you must include it in your answer.

```
@Test
public void testSameUpToEmpty() {
    BooleanCollection bc = new BooleanCollection();
    BooleanCollection other = new BooleanCollection();

    assertTrue(bc.sameUpTo(other, 0));
}
```

KEY:
[1] – two BooleanCollection objects correctly instantiated and assigned to variables of the right type
[2] – correct call to sameUpTo
[1] – correct call to assertTrue (or equivalent assertions)

[1] mark penalty applied for any additional assertions that are NOT correct.

(1.d) [6 points] Write a single test method that tests the `BooleanCollection`'s `sameUpTo` method when `this` and `other` have the same first 2 elements in the same order, but `other` has one extra element. Your test should check that `sameUpTo` produces the correct result when asked to determine if the first 2 elements in the collections are the same and in the same order. As in 1.c above, this method would be in, for example, a `BooleanCollectionTest` class. Assume that no `@Before` method exists - if you wish to make use of one, you must include it in your answer.

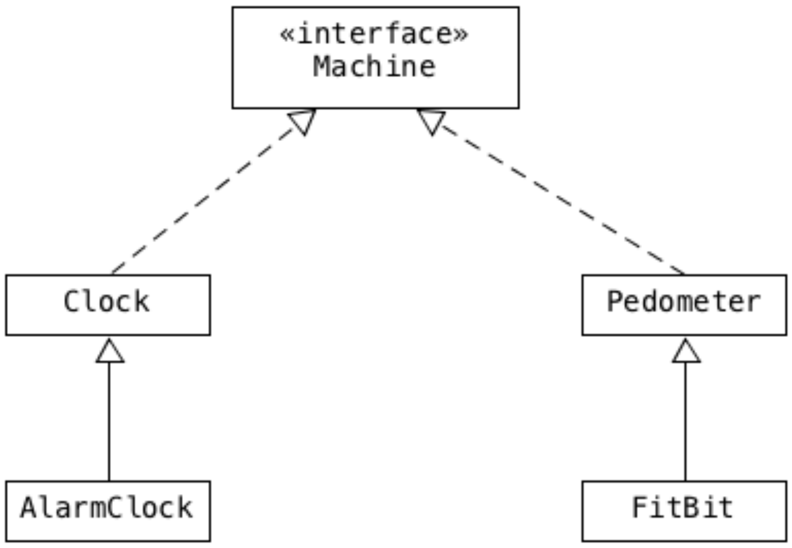
```
@Test
public void testSameUpToNonEmpty() {
    BooleanCollection bc = new BooleanCollection();
    BooleanCollection other = new BooleanCollection();

    bc.add(true);
    bc.add(false);
    other.add(true);
    other.add(false);
    other.add(true);

    assertTrue(bc.sameUpTo(other, 2));
}
```

KEY:
[2] – correctly construct a BooleanCollection object (1 mark) and add two elements (1 mark)
[2] – correctly construct another BooleanCollection object and add three elements (1 mark) the first two of which are the same as those in the other BooleanCollection object and in the same order (1 mark)
[2] – correct call to sameUpTo (1 mark) and assertTrue (or equivalent assertion) (1 mark)

[1] mark penalty applied for additional assertions that are NOT correct.

2. TYPE HIERARCHIES [6 points]

Draw the type hierarchy that includes all the types in the `model` package of the `Machines` project (the code is attached – note that it spans several pages). Just include type names and extends/implements relationships between types. Do NOT include methods or fields (the diagram will just get too big).



KEY:
[-1] each extra/missing type

[-1] for wrong arrows (solid/dotted) or arrowheads; unless solid line is used for implements and <<interface>> annotation is included or "implements" written alongside arrow.

[-1] for misplaced <<interface>> or <<abstract>> annotations.

[-1] for arrows consistently in wrong direction; [-2] if arrow directions are inconsistent.

3. APPARENT AND ACTUAL TYPES [16 points]
Assume the following variables have been created with the indicated types and have been correctly initialized. The initial value of these variables does not affect this question in any way. None of these variables has the value `null`.

```
Machine machine;
Clock clock;
AlarmClock alarmClock;
Pedometer pedometer;
FitBit fitBit;
```

(3.a) [12 points] Circle Yes or No indicating whether each of the following sets of statements will compile. For each one explain very briefly (do not exceed the provided space) why it will or will not compile.

```
machine = new Machine();
```
Will it compile? *Yes* ****** No ******
Explain:

You cannot create an instance of an interface.

```
machine = new Pedometer();
```
Will it compile? ****** Yes ****** *No*
Explain:

Pedometer is a subtype of Machine

```
machine = clock;
```
Will it compile? ****** Yes ****** *No*
Explain:

Clock is a subtype of Machine

```
clock = machine;
```
Will it compile? *Yes* ****** No ******
Explain:

Machine is not a subtype of Clock

```
pedometer.getGoal();
```
Will it compile? *Yes* ****** No ******
Explain:

There is no method getGoal specified for Pedometer

```
pedometer = new FitBit();
pedometer.getDistance();
```
Will these statements compile? ****** Yes ****** *No*
Explain:

FitBit is a subtype of Pedometer
The method getDistance is specified for Pedometer

(3.b) [4 points] Suppose there is a method with the following signature (note that the implementation is not important in the context of this question):

```
public void print(Pedometer p) {
    // stub
}
```

(3.b.i) [2 points] Will the following call to this method compile assuming it is made from within the class in which it is declared?

```
print(new Clock());
```

Will it compile? *Yes* *** ***No*** ***
Explain:

Clock is not a subtype of Pedometer

(3.b.ii) [2 points] Will the following call to this method compile assuming it is made from within the class in which it is declared?

```
print(new FitBit());
```

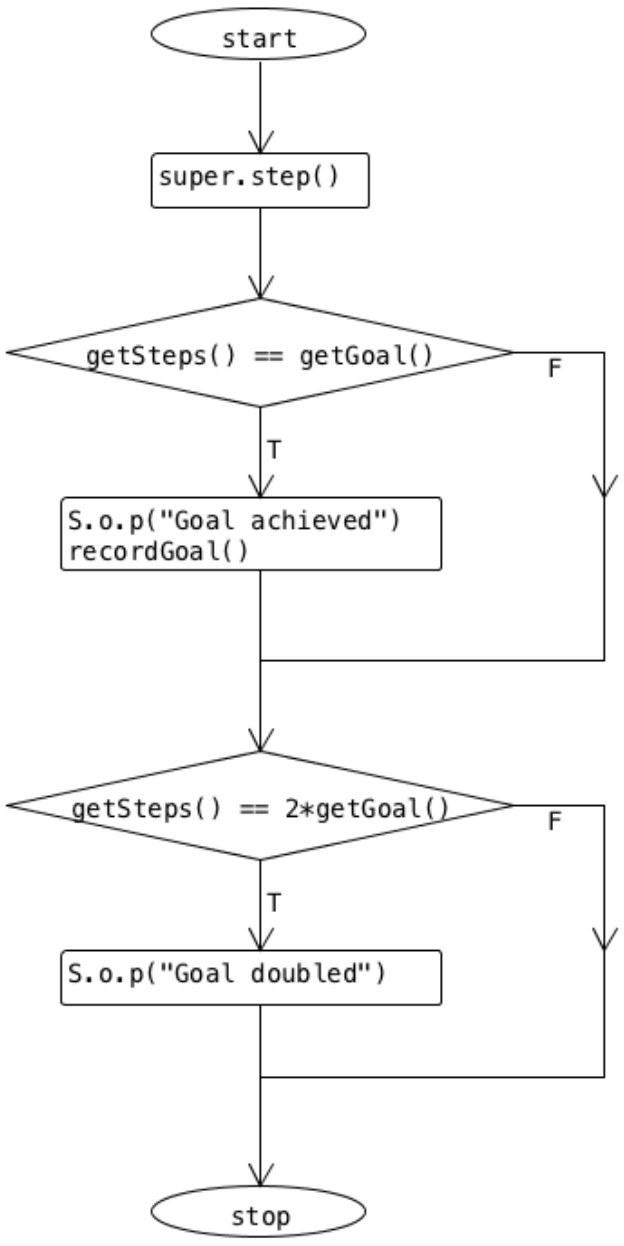
Will it compile? *** ***Yes*** *** *No*
Explain:

FitBit is a subtype of Pedometer

KEY:
[2] marks for each sub-part in 3a) and 3b)
  - [1] yes/no correct
  - [1] correct explanation (no marks for explanations that are not succinct or that do not capture the essence of the answer provided above)

4. CONTROL FLOW [18 points]

(4.a) [7 points] Draw an intra-method flowchart for the method `step` in the Class `FitBit`.



KEY:

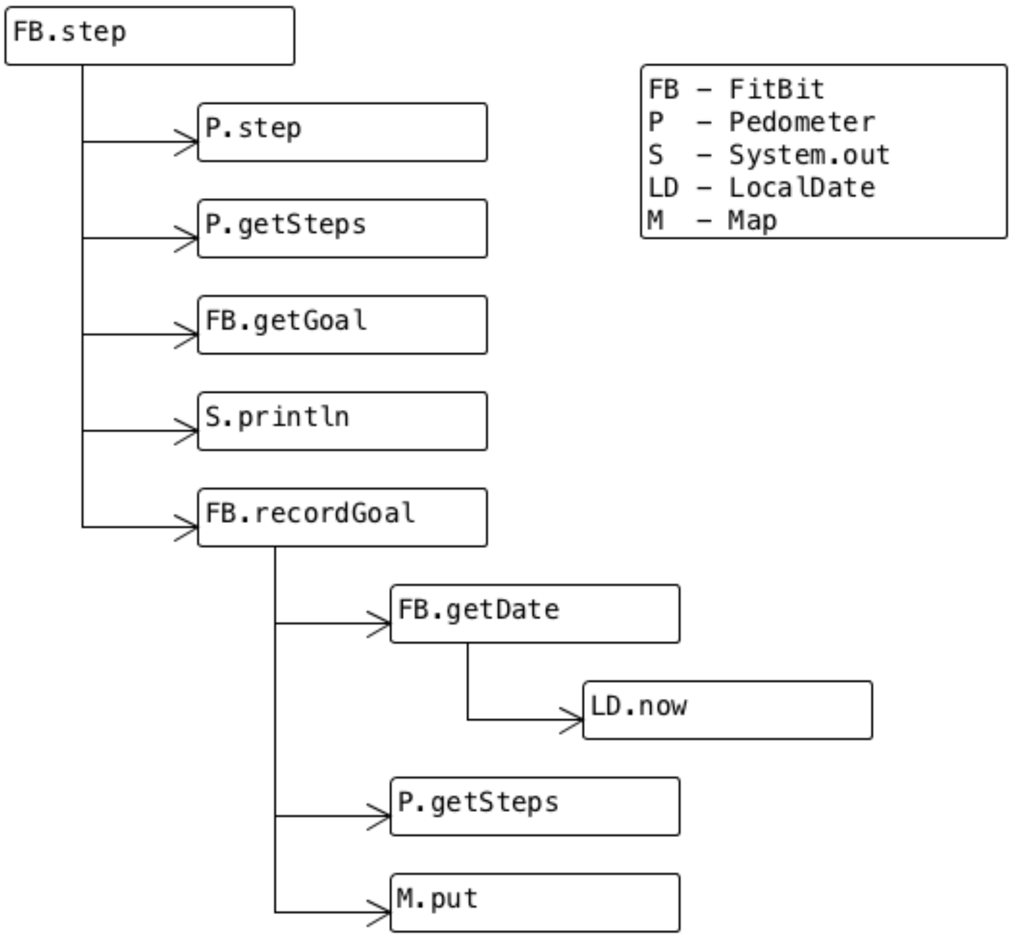
[3] – first branch or second (whichever is more correct)

- **[1] Boolean with two branches**
- **[1] correct true branch**
- **[1] correct false branch**

[3] – branches must be correctly positioned relative to each other – no part marks

[1] – T/F labels

(4.b) [7 points] Draw an inter-method call graph for the method `step` in the Class `FitBit`. Include the first level of calls to the Java library. Be sure to include a legend if you abbreviate type names.



KEY:
[-1] each missing method
[-1] each extra method (no deduction for duplicates called from same method)
[-1] each method not connected to correct parent in graph
[-1] wrong type specified on method call (applied at most once)

(4.c) [4 points] What is printed out when the following method runs:

```
public static void main(String[] args) {
   List<Pedometer> pedometers = new ArrayList<>();
   Pedometer p = new Pedometer();
   FitBit f = new FitBit();

   pedometers.add(p);
   pedometers.add(f);
   pedometers.add(p);

   for (Pedometer p : pedometers) {
      System.out.println(p.getName());
   }
}
```

Pedometer
FitBit
Pedometer

KEY:
[2] – first and last lines are Pedometer (no marks if only one line is correct)
[2] – second line is FitBit