FILTER

OBJECTIVES

This assignment focuses on abstract classes and interfaces. You must implement the Filter interface in over a dozen ways.

OVERVIEW

When looking for data on the Internet, you are often interested in only a subset of the data in a particular set. Perhaps you want to scan through a text file and find only email addresses or find only dollar amounts. Perhaps you are scanning through a list of Employee objects and only want to find the employees that are at least 50 years old. These goals can be accomplished by applying a Filter to objects where the Filter defines which elements are of interest and which elements are not interesting.

DETAILS

Consider the Filter interface shown below. The interface defines a single abstract method named ‘accepts’ that will determine whether the filter accepts an element or not. The method will return true if it accepts an element and false otherwise.

```java
public interface Filter<E> {
    public boolean accepts(E element);
}
```

You must write numerous implementations of the Filter interface. Each implementation must be in a separate file. These implementations are specified below and note that each of these methods is case insensitive.

NON GENERIC FILTERS

1. IsReddish. Accepts any java.awt.Color object such that the red channel is the greatest of the red, green, and blue channels. For example, new IsReddish() is a filter that accepts Color.red but not Color.yellow.
2. IsBright. Accepts any java.awt.Color object such that the sum of the red, green, and blue channels is at least 256. For example, new IsBright() is a filter than accepts Color.yellow but not Color.red.

GENERIC FILTERS

3. StartsWith. Accepts any element such that the textual representation starts with a given prefix. The constructor must accept a single string that specifies the prefix. For example, new StartsWith<String>("email") is a Filter accepting any String starting with “email”. As another example, new StartsWith<Contact>("email") is a Filter accepting any Contact object such that the toString() method produces a string containing “email”.
4. Contains. Accepts any element such that the textual representation contains a specified substring. The constructor must accept a single string that specifies the substring. For example, new Contains<String>("email") would be a Filter accepting any String containing "email".
5. LessThan. Accepts any element such that the textual representation is lexicographically less than a specified reference string. The constructor must accept a single string that specifies the reference string.
For example, new LessThan<String>("mom") would be a Filter accepting any Object such that the textual representation of that object occurs before “mom” alphabetically.

6. **AsLongAs.** Accepts any element such that the textual representation is at least N characters in length where N is an integer value given to the constructor. For example, new AsLongAs<String>(3) would be Filter accepting any Object such that the textual representation is at least three characters in length.

7. **IntegerNumber.** Accepts any element such that the textual representation is an integer number (as defined by the Java language specification). For example, new IntegerNumber<String>() would be a Filter accepting the strings “123”, “0x35” and “-12” but not accepting “12.3” or “352359935629696991353292593569392”.

8. **Censor.** Accepts any element such that the textual representation is not in a censored list. The constructor must accept a single parameter of type String[] that represents the list of censored words. For example, consider a simple list named ‘words’ that contains {"darn", "dang", "!@X?%$"}. We now construct a Filter as in new Censor<String>(words) that would accept all strings that are not one of the three censored strings.

9. **ContainsAll.** Accepts any element such that the string representation of that element contains each of the letters (in any order) of a reference list-of-letters. The constructor must accept a single String that specifies the list-of-letters. For example, new ContainsAll<String>("ob") is a Filter that accepts the strings “box” and “below” because each of these strings contain both an ‘o’ and a ‘b’.

10. **Not.** Accepts any element that is not accepted by a reference filter. The constructor must accept a single Filter that denotes the reference filter. For example, new Not<String>(new Contains<String>("email")) is a Filter accepting any String that does not contain “email”.

11. **All.** Accepts any element such that the textual representation of that element is accepted by a list of reference filters. The constructor must accept a List of Filters that denotes the reference filters. For example, assume that List<Filter<String>> refs = new ArrayList<Filter<String>>(). We can add several filters to this list and then construct an All filter as in new All<String>(refs). This is a filter that accepts any string that is accepted by each filter in refs.

12. **BinaryFilter.** This is an abstract class that implements Filter. The constructor must accept two Filter objects and we denote as the LEFT and RIGHT filter.

   a. **Or.** This non-abstract class is a BinaryFilter that accepts any element that is accepted by either the LEFT, the RIGHT or both the LEFT and RIGHT filters. For example, new OrFilter( new AsLongAs(20), new IntegerNumber() ) is a filter accepting any element having a textual representation that is either at least 20 characters long or is an integer number.

   b. **And.** This non-abstract class is a BinaryFilter that accepts any element that is accepted by both the LEFT and RIGHT filters. For example, new AndFilter( new Contains("."), new AsLongAs(10) ) is a filter accepting any element having a textual representation that contains a ‘.’ and is at least 10 characters long.

   c. **Xor.** This non-abstract class is a BinaryFilter that accepts any element that is accepted by exactly one of the LEFT and RIGHT filters. For example, new XorFilter( new Contains("1"), new IntegerNumber() ) is a filter that accepts any element having a textual representation that either contains 1 and is not an integer or that is an integer that doesn’t contain a 1.