AP® COMPUTER SCIENCE AB
2008 SCORING GUIDELINES

Question 4: Filter Objects (Design)

Part A:	OrFilter	**5 points**

+1/2 `class OrFilter implements Filter`

+1/2 declare `private` instance variable(s) capable of storing a collection of `Filter`s
Note: Two Filters suffice, as long as `add` builds and stores complex filters.

+1/2 constructor, `add`, `accept` headers correct
Note: If `add` does not return void, must return a legal value.

+1/2 constructor stores both parameters in instance variable(s) (initialize if necessary)

+1/2 `add` stores parameter in instance variable(s)

+2 1/2 `accept`

- **+1** access stored filters
  - **+1/2** correctly access a stored filter (if in a collection, must be in a loop)
  - **+1/2** access all stored filters (lose this if index out-of-bounds)
- **+1 1/2** determine whether to accept
  - **+1/2** call `accept(`*text*`)` on an accessed filter
  - **+1** return correct boolean value in all cases

Part B:	buildFilter	**4 points**

+1 access all elements of `desirable` (lose this if index out-of-bounds)

+1 construct `SimpleFilter` for each `desirable` element and `notAllowed`

+1 correctly construct all complex filters (must have at least one complex filter)

+1 build and return correct filter

Question 4: Filter Objects (Design)

PART A:

```
public class OrFilter implements Filter
{
  private ArrayList<Filter> filters;

  public OrFilter(Filter f1, Filter f2)
  {
    filters = new ArrayList<Filter>();
    add(f1);
    add(f2);
  }

  public void add(Filter f)
  {  filters.add(f);  }

  public boolean accept(String text)
  {
    for (Filter f : filters)
    {
      if (f.accept(text))
        return true;
    }
    return false;
  }
}
```

ALTERNATE SOLUTION:

```
public class OrFilter implements Filter
{
  private Filter filter1, filter2;

  public OrFilter(Filter f1, Filter f2)
  {
    filter1 = f1;
    filter2 = f2;
  }

  public void add(Filter f)
  {
    filter1 = new OrFilter(filter1, filter2);
    filter2 = f;
  }

  public boolean accept(String text)
  {  return filter1.accept(text) || filter2.accept(text);  }
}
```

Question 4: Filter Objects (Design) (continued)

PART B:

```
public static Filter buildFilter(String[] desirable, String notAllowed)
{
  OrFilter orF = new OrFilter(new SimpleFilter(desirable[0]),
                              new SimpleFilter(desirable[1]));
  for (int i = 2; i < desirable.length; i++)
  {
    orF.add(new SimpleFilter(desirable[i]));
  }
  return new AndFilter(orF, new NotFilter(new SimpleFilter(notAllowed)));
}
```

Write the `OrFilter` class below.

AB4a 1

```
public class OrFilter impements Filter {
        private ArrayList<Filter> filters;
        public OrFilter(Filter f1, Filter f2) {
                filters. = new ArrayList <Filters>();
                filters.add(f1);
                filters.add(f2);
        }
        public boolean accept(String s) {
                boolean result = false;
                for(Filter f:filters) {
                        result = !result && !f.accept(s);
                }
                return !result;
        }
        public void add(Filter f) {
                filters.add(f);
        }
}
```

Part (b) begins on page 22.

GO ON TO THE NEXT PAGE.

Complete method buildFilter below.

AB4a$_2$

```
/** @param desirable contains strings that are allowed
 *          Precondition: desirable.length > 1
 *  @param notAllowed the string that is not allowed
 *  @return a Filter that accepts strings that contain at least one string
 *           in desirable and do not contain notAllowed.
 */
public static Filter buildFilter(String[] desirable,
                                 String notAllowed) {
        Filter ret = new Simple.Filter(desirable[0]);
        int i = 1;
        while(i< desirable.length){
                ret = new OrFilter(ret, new SimpleFilter(desirable[i]));
                i++;
        }
        ret = new AndFilter(ret, new NotFilter(new SimpleFilter(notAllowed)));
        return ret;
}
```

GO ON TO THE NEXT PAGE.

Write the `OrFilter` class below.

AB4b1

```
public class OrFilter implements Filter {
    ArrayList<Filter> filters;
     public OrFilter (Filter a, Filter b) {
          filters add(a);
          filters.add(b);

     }

     public boolean accept (String text) {
          ArrayList<boolean> acceptResults = new ArrayList<boolean
          Iterator itr = filters.iterator();
            while (itr hasNext()) {
                  Filter current Filter = itr.next();
                accept Results. add(current Filter. accept(text))}

            }

          Iterator itr 2 = acceptResults.iterator();
            while (itr2 hasNext()) {
                  booleam currentResult = itr 2. next();
                  if(current Result)
                        return true;
              }
     }

    public void add (Filter filter){

        filters. add(filter);

    }

}
```

Part (b) begins on page 22.

GO ON TO THE NEXT PAGE.

Complete method `buildFilter` below.

AB4b2

```
/** @param desirable contains strings that are allowed
 *          Precondition: desirable.length > 1
 *  @param notAllowed the string that is not allowed
 *  @return a Filter that accepts strings that contain at least one string
 *           in desirable and do not contain notAllowed.
 */
public static Filter buildFilter(String[] desirable,
                                 String notAllowed)
{  SimpleFilter firstFilter = new SimpleFilter(desirable[0]);
   SimpleFilter secondFilter = new SimpleFilter(desirable[1]);
    OrFilter  desirableFilter = new OrFilter(firstFilter, secondFilter
      for(int i = 2; i < desirable.size(); i++)
          desirableFilter.add(new SimpleFilter(desirable[i]);
     NotFilter notAllowedFilter = new NotFilter(notAllowed);
        Filter finalFilter = new AndFilter(desirableFilter,
                                            notAllowedFilter)

     return finalFilter;

}
```

Write the `OrFilter` class below.

AB4C1

```
public class OrFilter implements Filter
{     private ArrayList <Filter> myFilters;
    public OrFilter (Filter a , Filter b)
    {
        ArrayList <Filter> myFilter = new ArrayList <Filter> (a, b);
    }
    public boolean accept (String aString)
    {
        if ( myFilters.get (0).accept (aString) || myFilters.get(1).accept (aString)
            return true;

        return false;
    }
    public void add (Filter aFilter)
    {
        myFilters.add (aFilter);
    }
}
```

Part (b) begins on page 22.

GO ON TO THE NEXT PAGE.

AB 4 c 2

Complete method `buildFilter` below.

```
/** @param desirable contains strings that are allowed
 *          Precondition: desirable.length > 1
 *  @param notAllowed the string that is not allowed
 *  @return a Filter that accepts strings that contain at least one string
 *            in desirable and do not contain notAllowed.
 */
public static Filter buildFilter(String[] desirable,
                                 String notAllowed)
{
        AndFilter combine = null;
          combine.add ( OrFilter ( desirable ));
          combine.add (NotFilter ( not Allowed);
          myFilters. add( combine );

}
```

GO ON TO THE NEXT PAGE.

Question 4

Overview

This question focused on designing and implementing a class, then using that class to perform a particular task. The `Filter` interface was provided, along with a description of how a `SimpleFilter` class would behave. In part (a) students were required to design and implement another `Filter` class, demonstrating their knowledge of how instance variables, constructors, and methods together make a class. In part (b) students had to write code that constructed multiple `Filter` objects and combined them to compute a particular Boolean function.

Sample: AB4a
Score: 8

The student writes a nearly perfect solution and demonstrates a good understanding of the Java programming language, object-oriented design, and algorithmic thinking. However, the student inverts the logic required for the solution in one place.

Part (a):

The body of the `accept()` method is faulty. The method will not always return the correct value. Suppose that the `OrFilter` has four components. When a caller invokes the `accept()` method of an `OrFilter`, the object will loop to determine whether or not each of its four component filters accepts the given string. Suppose further that in a particular case the first and last component filters reject the string, but the middle two filters accept it. Then here is what happens in the loop:

Iteration #1	1st component filter rejects string	`result` is set to `true`
Iteration #2	2nd component filter accepts string	`result` is set to `false`
Iteration #3	3rd component filter accepts string	`result` is set to `false`
Iteration #4	4th component filter rejects string	`result` is set to `true`

When the method exits from the loop, the method returns the logical complement of `result` to its caller. In this case, it returns "not true." It should return "true" because at least one of its components (in fact, two components in this case) accepted the string.

Part (b):

The student understands that the specification calls for the construction of complex filters from simple filters. The constructor for the `SimpleFilter` class requires its caller to supply a `String`. The constructors for the `AndFilter`, `NotFilter`, and `OrFilter`s require their callers to supply `Filter`s.

First, the student constructs a `SimpleFilter` with the first element of the `desirable` array and assigns it to a `Filter` variable `ret`. The student then traverses the rest of the `desirable` array and repeatedly updates `ret` with a new `OrFilter` constructed with the current value of `ret` and a `SimpleFilter` created from the next element of the `desirable` array. After all the elements of the `desirable` array have been incorporated into the filter, the student creates an `AndFilter` using `ret` and a `NotFilter` (constructed with a `SimpleFilter` made with the parameter `notAllowed`) and assigns the result to `ret`. The student then returns `ret`.

Question 4 (continued)

Sample: AB4b
Score: 6

Part (a):

The student does not make the instance variable private, so the ½ point allocated for the declaration of an instance variable was not earned. The student does not instantiate the list before trying to add to it. For this reason, the student did not earn the ½ point allocated for composing the body of the constructor.

The student provides a means by which the `accept()` method can return "true" to its caller but provides no means by which the method can return "false." So the student did not earn the 1 point allocated for returning a correct value to the caller.

Part (b):

The student passes a string to the constructor for the `NotFilter` class. The constructor for that class requires another filter. A correct solution passes the `notAllowed` string to the constructor for the `SimpleFilter` class, then passes a reference to the `SimpleFilter` to the constructor for the `NotFilter` class. For this reason, the student did not earn the 1 point allocated for building `SimpleFilter`s for each of the strings passed to the `buildFilter()` method.

Sample: AB4c
Score: 3

Part (a):

The student gives the class the right number and types of elements, but the student does not include the right logic within the bodies of all of the class' methods.

The student received 3 points by earning ½ point in each of six places. The student did not earn 2 of the points allocated for writing the class in part (a) of this problem. The student earned ½ point for writing the class header correctly; ½ point for declaring a private instance variable that is a reference to a kind of Collection; and ½ point for writing the headers for the constructor, the `add()` method, and the `accept()` method correctly. The student earned ½ point for writing the body of the `add()` method correctly; ½ point for accessing at least one of the `OrFilter`'s component filters in the `accept()` method; and ½ point for calling a component's `accept()` method within the `OrFilter`'s `accept()` method.

The student did not earn the ½ point allocated for writing the body of the constructor correctly because the student redeclares (shadows) the instance variable and attempts to assign references using a nonexistent two parameter constructor for `ArrayList`.

The student did not earn the ½ point allocated for accessing all component filters in the `accept()` method. A loop that visits each element of the object's `ArrayList` would have solved this part of the problem.

The student did not earn the ½ point for returning from the `accept()` method a correct value in all cases. The method only works when the `OrFilter` has exactly two component filters.

Question 4 (continued)

Part (b):

The student did not earn any of the 4 points allocated for writing the body of the `buildFilter()` method.