Final Exam

EE363 - Fall 2015
Software Components and Generic Programming

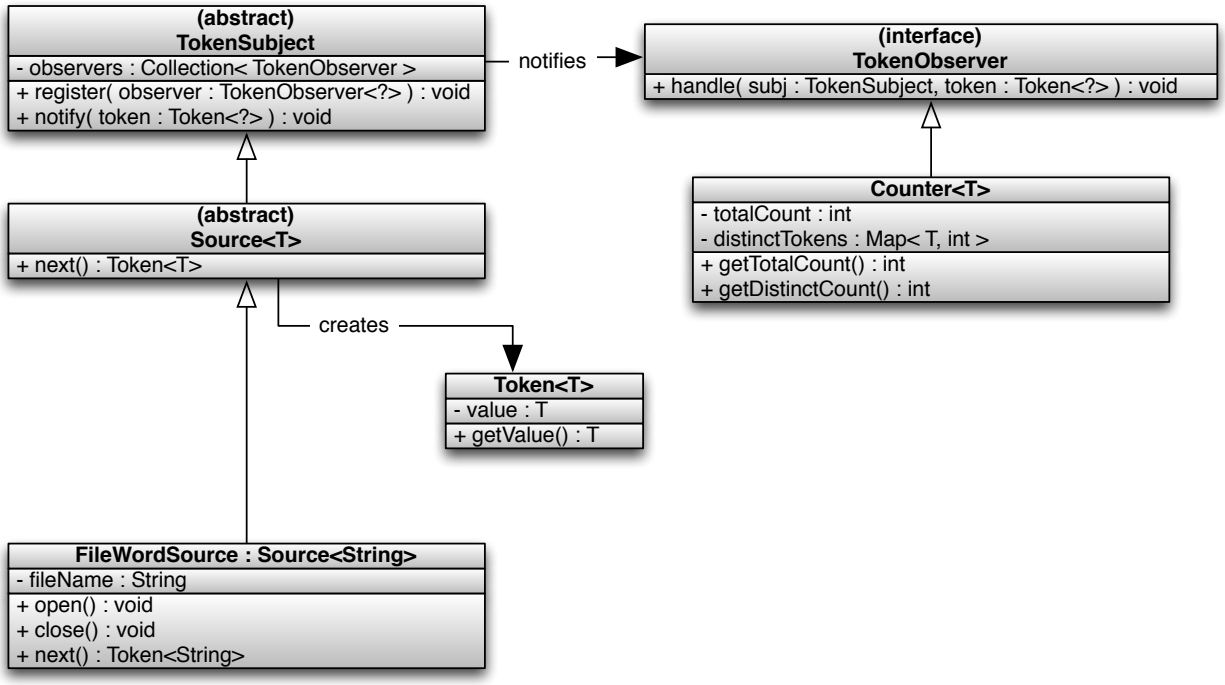
Instructions

Please read each question carefully, and answer each in the space provided. Use the back side of the pages if you need additional space, and be sure to indicate clearly which questions you are answering if you do.

_____

Name

Counter Framework



The UML diagram depicts a model for the counter application framework. The source code listing for each class is contained at the end of this exam.

Answer the following questions about the design and implementation of this word counter system.

Problem 1:
Describe which design patterns are used in this implementation. For each design pattern used, describe the classes that participate in that pattern, and what role each class plays to fulfill the pattern's responsibilities.

Problem 2: The following code block *compiles perfectly fine.*

```
public static final void main( String[] args ) {
    Counter<Integer> stringCounter = new Counter<Integer>();

    FileWordSource fws = new FileWordSource( "/usr/share/dict/words" );
    fws.register( stringCounter );

    long start = System.currentTimeMillis();
    while ( fws.next() != null ) {
        ;
    }

    System.out.println( "Got " + stringCounter.getTotalCount()
        + " words from source." );
}
```

List any problems you can think of with this code block; pay particular attention to the parameterized ("generic") types.

NOTE: this code also runs just fine... for +5 extra credit on this exam, explain why.

Problem 3: What is the Big-O runtime of the program given an arbitrary input file? Explain how you came up with that value...

Problem 4: What would the Big-O runtime of the program, given an arbitrary input file, be, had I implemented counter like this instead? Explain...

Hypothetical Counter.java

```
package com.timfanelli.fall2012.ee363.counter;

import java.util.LinkedList;
import java.util.List;

public class Counter<T> implements TokenObserver<T> {
    private int totalCount = 0;

    private List<Token<T>> distinctTokens =
            new LinkedList<Token<T>>();

    public void addToken( Token<T> token ) {
        ++totalCount;

        if ( ! distinctTokens.contains(token) ) {
            distinctTokens.add( token );
        }

    }

    public int getTotalCount() { return totalCount; }
    public int getDistinctCount() { return distinctTokens.size(); }

    @Override
    public void handle(TokenSubject tokenSubject, Token<T> token) {
        this.addToken(token);
    }
}
```

Problem 5: The author of the FileWordCounter needs to be mindful of the fact that the definition of a word may change. His default implementation uses a regular-expression pattern to match on “non-word characters” (\W); however, more complicated document types, such as scientific documents or software specifications, may have special symbols that are parts of words that wouldn’t otherwise be recognized.

What can the author do to represent and isolate this behavior in a manner that would allow him (or anyone else) to add new word definitions later?

Counter.java

```
package com.timfanelli.fall2012.ee363.counter;

import java.util.HashMap;
import java.util.Map;

public class Counter<T> implements TokenObserver<T> {
    private int totalCount = 0;
    private Map<Token<T>, Integer> distinctTokens =
            new HashMap<Token<T>, Integer>();

    public void addToken( Token<T> token ) {
        ++totalCount;

        if ( distinctTokens.containsKey( token ) ) {
            Integer count = distinctTokens.get( token );
            distinctTokens.put( token, new Integer( count.intValue() + 1 ) );
        }
        else {
            distinctTokens.put( token, 1 );
        }

    }

    public int getTotalCount() { return totalCount; }
    public int getDistinctCount() { return distinctTokens.keySet().size(); }

    @Override
    public void handle(TokenSubject tokenSubject, Token<T> token) {
        this.addToken( token );
    }
}
```

Source.java

```
package com.timfanelli.fall2012.ee363.counter;

public abstract class Source<T> extends TokenSubject {
    public abstract Token<T> next();
}
```

TokenObserver.java

```
package com.timfanelli.fall2012.ee363.counter;

public interface TokenObserver<T> {
    void handle(TokenSubject tokenSubject, Token<T> token);
}
```

TokenSubject.java

```
package com.timfanelli.fall2012.ee363.counter;

import java.util.ArrayList;
import java.util.List;

public abstract class TokenSubject {
    private List<TokenObserver> observers = new ArrayList<TokenObserver>();

    public void register( TokenObserver<?> observer ) {
        observers.add( observer );
    }

    public void notify( Token<?> token ) {
        for ( TokenObserver observer : observers ) {
            observer.handle( this, token );
        }
    }
}
```

Token.java

```
package com.timfanelli.fall2012.ee363.counter;

public class Token<T> {
	private T element;

	public Token( T data ) {
		this.element = data;
	}

	public T getElement() {
		return element;
	}
}
```

FileWordSource.java

```
package com.timfanelli.fall2012.ee363.counter;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileWordSource extends Source<String> {
	private Scanner inputScanner;

	public FileWordSource( String filename ) {
		try {
			inputScanner = new Scanner( new FileInputStream( filename ) );
			inputScanner = inputScanner.useDelimiter("\\W");

		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
	}

	@Override
	public Token<String> next() {
		String nextValue = null;
		if ( inputScanner.hasNext() )
		{
			nextValue = inputScanner.next();
			Token<String> token =  new Token<String>(nextValue);
			notify( token );
			return token;
		}

		return null;
	}

}
```

Main.java

```
package com.timfanelli.fall2012.ee363.counter;

public class Main {
    public static final void main( String[] args ) {
        Counter<String> stringCounter = new Counter<String>();
        FileWordSource fws = new FileWordSource( "/usr/share/dict/words" );
        fws.register( stringCounter );

        while ( fws.next() != null ) {
            ;
        }

        System.out.println( "Got " + stringCounter.getTotalCount() +
            " words from source." );
    }
}
```