Assignment 4 – Wei Sun, May 8th

1. What's the difference between final, finally? What is finalize()?

final: final is a reserved keyword in Java, it is a non-access modifier, which can be used on class, method, and variable

- final class cannot be inherited by subclass
- final method cannot be overridden
- final variable cannot be re-referenced, and it must be initialized before the constructor

finally: finally is a reserved keyword in Java, it is used for exception handling, normally placed after try…catch block. The finally block will always be executed no matter if the exception is handled or not

finalize: finalize is a method for garbage collection, it performs cleanup before objects are being destroyed

2. What's the difference between throw and throws?

The throws keyword is used in a method signature and declares which exceptions can be thrown from a method.

The throw keyword is used within a method body, or any block of code, and is used to explicitly throw a single exception.

3. What are the two types of exceptions?

Checked exceptions and unchecked exceptions.

Checked exceptions are also known as compile-time exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error. For example, SQLException, IOException, and ClassNotFoundException.

Unchecked exceptions are those exceptions that occur during the execution of the program. They are also referred to as Runtime exceptions. These exceptions are generally ignored during the compilation process. They are not checked while compiling the program. For example, NullPointerException, ArrayIndexOutOfBoundException, ClassCastException.

4. What is error in java?

In Java, an error is a subclass of Throwable that tells that something serious problem is existing, and a reasonable Java application should not try to catch that error. Generally, it has been noticed that most of the occurring errors are abnormal conditions and cannot be resolved by normal conditions.

Error cannot be handled by try…catch block like Exception. So, when an error occurs, it causes termination of the program.

5. Exception is object, true or false?

True, Exceptions are objects in Java. Exceptions are object instantiated from classes which extends throwable.

6. Can a finally block exist with a try block but without a catch?

Yes, it is possible to have a try block without a catch block by using a final block, as a final block will always execute even there is an exception occurred in a try block

7. From java 1.7, give an example of the try-resource feature.

Try-with-resources allows us to declare resources to be used in a try block with the assurance that the resources will be closed after the execution of that block.

Example:

```
try (PrintWriter writer = new PrintWriter(new File("test.txt"))) {
    writer.println("Hello World");
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

8. What will happen to the Exception object after exception handling?

After the Exception object is handled, it will be garbage collected in the next garbage collection.

9. Can we use String as a condition in switch(str){} clause?

Yes, String can be used as condition in switch after Java 1.7, it will call String.equals() method to compare the expressions.

10. What's the difference between ArrayList, LinkedList and Vector?

All ArrayList, LinkedList, and Vectors implement the List interface. Both ArrayList and Vectors use dynamically resizable arrays as their internal data structure. Vector is synchronized, while both ArrayList and LinkedList are non-synchronized.

ArrayList and Vector use dynamic arrays to store its elements and maintains insertion order, while LinkedList used doubly linked list for implementation. The get and set operations in ArrayList and Vector has time complexity of O(1), while O(n) for LinkedList. The insertion operation has time complexity of O(n) in ArrayList and Vector, while O(1) for LinkedList.

11. What's the difference between hashTable and hashMap?

HashMap is non-synchronized, while HashTable is synchronized, so HashMap is more efficient. HashMap allows Null key, while HashTable does not.

12. What is static import?

The static import declaration is analogous to the normal import declaration. Where the normal import declaration imports classes from packages, allowing them to be used without package qualification, the static import declaration imports static members from classes, allowing them to be used without class qualification.

13. What is static block?

A static block can be used for static initialization of a class. This code inside the static block is executed only once: the first time the class is loaded into memory.

14. Explain the keywords:

default(java 1.8): Specifies the default block of code in a switch statement

break: Breaks out of a loop or a switch block

continue: Continues to the next iteration of a loop

synchronized: A non-access modifier, which specifies that methods can only be accessed by one thread at a time

strictfp: Restrict the precision and rounding of floating point calculations to ensure portability

transient: A non-accesss modifier, which specifies that an attribute is not part of an object's persistent state. Variables with transient modifier won't be saved in serialization.

volatile: Indicates that an attribute is not cached thread-locally, and is always read from the main memory

instanceOf: Checks whether an object is an instance of a specific class or an interface

15. Create a program including two threads – thread read and thread write.
Input file ->Thread read -> Calculate -> buffered area
Buffered area -> Thread write -> output file
Detailed description is in assignment4.txt file. Sample input.txt file.
Attached files are input.txt and a more detailed description file.

```java
public class threadReadAndWrite {

    public static void main(String[] args) {
        BlockingQueue<String> queue = new ArrayBlockingQueue<String>(1024);

        ReaderThread reader = new ReaderThread(queue);
        WriterThread writer = new WriterThread(queue);

        (new Thread(reader)).start();
        (new Thread(writer)).start();
    }
}



class ReaderThread implements Runnable{
    protected BlockingQueue<String> blockingQueue = null;

    public ReaderThread(BlockingQueue<String> blockingQueue){
        this.blockingQueue = blockingQueue;
    }

    public String calculate(String s) {
        int res = 0;
        boolean add = true;
        String[] arr = s.split(" ");
        for(String ele : arr) {
            if(ele.equals("+")) {
                add = true;
            } else if(ele.equals("-")) {
                add = false;
            } else if(ele.equals("")) {
                continue;
            } else {
                if(add)
                    res += Integer.parseInt(ele);
                else
                    res -= Integer.parseInt(ele);
            }
        }
        return String.valueOf(res);
    }

    @Override
    public void run() {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(new
File("src/test/input.txt")));
            String buffer =null;
            while((buffer=br.readLine())!=null){
                if(!buffer.equals(""))
                    buffer = buffer + " = " + calculate(buffer);//empty line
                blockingQueue.put(buffer);
            }
            blockingQueue.put("EOF");  //When end of file has been reached

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch(InterruptedException e){
        }finally{
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

class WriterThread implements Runnable{
```

```java
    protected BlockingQueue<String> blockingQueue = null;

    public WriterThread(BlockingQueue<String> blockingQueue){
        this.blockingQueue = blockingQueue;
    }

    @Override
    public void run() {
        PrintWriter writer = null;

        try {
            writer = new PrintWriter(new File("src/test/outputFile.txt"));

            while(true){
                String buffer = blockingQueue.take();
                //Check whether end of file has been reached
                if(buffer.equals("EOF")){
                    break;
                }
                writer.println(buffer);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch(InterruptedException e){
        }finally{
            writer.close();
        }
    }
}
```