Test 1 – Version 1

Last, First Name		Student ID	

Question 1: Theory Questions [10 points in total]

List all the engineering benefits of Simplicity. [5 points]

> Five things -- easier to understand the code, easier to debug code, easier for code to run correctly, easier to test/validate, easier to modify the code.

List all the engineering benefits of Optimality. [5 points]

> Four things -- Fast code, minimal use of memory, low use of other computing resources, fewest lines of code

Question 2: Well-designed object 1 [20 points]

Do not write comments, but create a well-design object in every other way. This question asks you to implement a stack of human ages. The stack holds a maximum of 100 ages. You are only writing the stack class. Assume there is an unsafe calling environment accessing your class. A valid human age is an integer from 0 to 120. The class provides the following public methods: push (add an age to the top of stack), pop (remove and return an age from the top of stack), peek (do not remove age but return the age from the top of stack). You can write supporting methods. Write this class and all its members using all the (as needed) good design principles we saw in class. This is a Java programming question.

```
class Stack {
   private int array[];
   private int top;

   Stack() { array=new int[100];  top=0; }

   private boolean isFull() {
      if (top>99) return true; else return false;
   }

   private boolean isEmpty() {
      if (top <= 0) return true; else return false;
   }

   private boolan isValidAge(int n) {
      if (n >= 0 && n <= 120) return true; else return false;
   }
```

```
   public void push(int n) throws StackFullError, InvalidAgeError {
      if (isFull())       throw StackFullError;
      if (!isValidAge(n)) throw InvalidAgeError;

      array[top] = n;
      top++;
   }

   public int peek() throws StackEmptyError {
      if (isEmpty()) throw StackEmptyError;

      return array[top-1];
   }

   public pop() throws StackEmptyError {
      if (isEmpty()) throw StackEmptyError;

      top--;
      return array[top];
   }
}
```

Question 3: Well-designed object 2 [20 points]

A programmer wants to create a tool that can stack any object and compare the equality of any two stacks. Each stack can contain any object type. This means that objects currently in the stack may not be from the same class, for example a stack may contain House, Car, Dog objects. The programmer does not want to use the class Object in the solution.

The class is called StackTool. It implements both the stack and the compare two stacks code. All stackable objects must contain the method String getKey() that returns a string containing information about the object. This information will be used to compare the equality between two objects. The compare method is static and takes two objects as parameters. When comparing the equality of two stacks the class contains a method that checks if every element in each stack is the same, however it is not knowable what class type the user will put in the stack. The user of StackTool will need to write the compare method that the compare stack method will call. This will use the getKey() method on each object, returning true when the objects are the same and false when they are not. The same, means, that the string returned by getKey() has a specific rule important to the developer/user expressed in the compare methods they provide.

Do not write this entire program. Do not use English (French) to explain how you would do it. Write code snippets. Be complete. Show off the techniques you used to make your solution well-designed. This is a Java programming question.

Basic idea: StackTool uses two arrays which are stacks
Better idea: Students could have used two Stack classes instead

Strategy:

- All stackable objects must implement a common Interface. The stacks/arrays are of type Interface. Since the arrays and the objects implement the same interface, then the objects can be added to the array.
- The interface methods are: getkey() and a compare() method (either by extending a comparable interface or by just stating your own compare() method)
- StackTool has a static method that takes in two arrays or stacks as arguments, and iterates through the stacks comparing the contents of every element. This static method uses the compare() method from each object. The compare() method uses the getkey() method to extract the element it needs to compare.

Above is the solution in words. The student is required to write code snippets that show how they would do the above in Java code.

Max full points (or lower) for showing the correct code

Max 75% points (or lower) for showing partial code and partial text

Max 50% points (or lower) for just explaining without code