**CS 1302 – Sample Test 4 Handout**

*Tear this page off to use for reference* ● *Turn this page in with test.*

**Use this information for problems 1-3:**



**Use this information for problem 4 & 8:**



Class diagram for *Collection* classes on the reverse side



|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 4** | **Name:** |  |
| No electronics, phones, smart watches, notes, *etc.* allowed. |  | Print: FirstName LastName |

1. Reference the handout…
2. (10 points) A *Product* has an *id* and an inventory level, *inv.* Write the *PremiumProduct (PP)* constructor that accepts an *id* and an inventory level, *inv*. The *id* is required to be less than 9000 and the inventory level must be positive; otherwise, it should throw an exception.
3. (10 points) Write a snippet of code that (a) creates a *PremiumProduct (PP)* with a an *id* of 89 and an inventory (*inv)* of -30.00,(b) prints any exception that might be thrown.
4. (10 points) Consider the *Product* class on the handout. Write a complete *Comparator* class such that *Product* objects are ordered on their *id.*
5. (4 points) Suppose you have a list, *products,* of *Product* objects*.* Write a snippet of code to use the comparator to sort *products* on their *id* ascending.
6. (4 points) Write either 1 or 2 lines of code to create the appropriate type of *Collection* class that could hold *Product* objects ordered on their *id*. In other words, whenever you add a new *Product* to this collection, it is put in the correct position.
7. (15 points) Write a method, *remHigh* that accepts a list of *Product* objects, and an inventory level, *max*. This method should remove the *Products* whose inventory is greater than *max* while also returning the removed accounts in a new list.
8. (30 points) Consider the classes shown on the handout. Notice that the *GoldAccount* has two constructors, one that takes an interest rate and one that does not. The one that doesn’t take an interest rate simply sets the interest rate to the default value, 0.05. Next, consider an example text file shown on the handout which contains information for a number of accounts. Write this static method below to read such a file and return a *Bank* object populated with the appropriate *Accounts*. **Note: (a) Assume all data is valid, (b) You do not need *try/catch*, (c) You may use: *I.pI(val)* to parse a string as an int.**

**public** Bank readAccounts(File file) **throws** FileNotFoundException {

 Scanner in = **new** Scanner(file);

}

1. (10 points) Do Exercises 1-4, Ch. 8, Sec. 2.1
2. (7 points) Do Exercises 11 & 14, Ch 8, Sec 9.1.
3. (7 points) What does this code output? There are some more examples in Ch. 6.

**public** **static** **void** main(String[] args) {

 *myMethod*(4,3); System.***out***.println();

 *myMethod*(4,0); System.***out***.println();

}

**public** **static** **void** myMethod( **int** x, **int** y ) {

 **try** {

 System.***out***.println("1");

 *divide*(x,y);

 System.***out***.println("2");

 }

 **catch**( Exception e ) {

 System.***out***.println("3");

 }

 **finally** {

 System.***out***.println("4");

 }

 System.***out***.println("5");

}

**public** **static** **void** divide( **int** x, **int** y ) {

 **int** z = x/y;

}

1. Extra Credit – See Chapter 2, Section 4 except there the text uses an array to hold the stack. In class, we used an ArrayList.
	1. Consider the partial *StackOfAccounts* class below. In the box below, write the *push, pop, & peek* methods for this stack.

**public** **class** StackOfAccounts {

 **private** ArrayList<Account> accounts;

 **private** **int** size = 0;

 **public** StackOfPersons() {

 accounts = **new** ArrayList<>();

 }

 **public** **boolean** isEmpty() {

 **return** size == 0;

 }

|  |
| --- |
|  |

}

* 1. Suppose you have two *Account* objects, *a1* and *a2.* Write a snippet of code to
		1. create a *StackOfAccounts* (abbreviate *SOA*),

|  |
| --- |
|  |

* + 1. push the two accounts onto the stack,

|  |
| --- |
|  |

|  |
| --- |
|  |

* + 1. pop the stack, storing the value that is returned in a variable.

|  |
| --- |
|  |