|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 4** | **Name:** | **KEY** |

**public** PremiumProduct(**int** id, **int** inventory) {

 **super**(id, inventory); // Must be first statement

 **if**(id>=9000 || inventory<0) {

 **throw** **new** IllegalArgumentException("id or inv invalid");

 }

}

PremiumProduct pp;

**try** {

 pp = **new** PremiumProduct(89, -30);

 System.***out***.println(pp);

}

**catch**(IllegalArgumentException e) {

 System.***out***.println(e);

}

**public** **class** ProductIdComparator **implements** Comparator<Product> {

 **public** **int** compare(Product p1, Product p2) {

 **return** p1.getId() - p2.getId();

 }

}

Collections.*sort*(products, **new** ProductIdComparator());

TreeSet<Product> productsOrdered = **new** TreeSet<>(**new** ProductIdComparator());

**public** **static** ArrayList<Product> remHigh(ArrayList<Product> products, **int** max) {

 ArrayList<Product> highInventory = **new** ArrayList<>();

 Iterator<Product> iter = products.iterator();

 **while**(iter.hasNext()) {

 Product p = iter.next();

 **if**(p.getInventory() > max) {

 iter.remove();

 highInventory.add(p);

 }

 }

 **return** highInventory;

}

1. To see solution: (a) go to Ch 7, Sec 7.4, Exercise 2 in the text, (b) there it will tell you the **exact package name** where the solution is found, (c) download & unzip the code samples for the Ch 7 text, (d) there you will find the package name, and inside there will be a class with then method. I don’t provide the package name here so that it is only in one place, in case it changes.
2. To see solution: (a) go to Ch 8, Sec 2.1, Exercises 1-4. (b) there it will tell you the **exact package name** where the solution is found, (c) download & unzip the code samples for the Ch 8 text, (d) there you will find the package name, and inside there will be a class with then method. I don’t provide the package name here so that it is only in one place, in case it changes.
3. To see solution: (a) go to Ch 8, Sec 9.1, Exercises 11 & 14. (b) there it will tell you the **exact package name** where the solution is found, (c) download & unzip the code samples for the Ch 8 text, (d) there you will find the package name, and inside there will be a class with then method. I don’t provide the package name here so that it is only in one place, in case it changes.

*myMethod*(4,3): 1, 2, 4, 5

*myMethod*(4,0): 1, 3, 4, 5

* 1. Consider the partial *StackOfPersons* class below. In the box below, write the *push* & *pop* methods for this stack.

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

 **private** ArrayList<Account> accounts;

 **public** StackOfPersons() {

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

 }

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

 **return** accounts.size() == 0;

 }

|  |
| --- |
| **public** **void** push(Account a) { accounts.add(a);}**public** Account pop() { **if**(!isEmpty()) { **return** accounts.remove(accounts.size-1); } **return** **null**;}**public** Account peek() { **if**(!isEmpty()) { **return** accounts(accounts.size-1); } **return** **null**;} |

}

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

|  |
| --- |
| StackOfAccounts soa = **new** StackOfAccounts(); |

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

|  |
| --- |
| sop.push(a1); |

|  |
| --- |
| sop.push(a2); |

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

|  |
| --- |
| Account a = soa.pop(); |