

CPSC 211

PRACTICE EXERCISES II

Note:

These questions are intended to help you practice and review the course material. Do not consider these questions as typical questions for the final; in particular, many could be difficult to finish within the length of the exam.

This part has exercises on the topics covered after the midterm. For topics covered before the midterm, study the exercises posted before the midterm, especially those for Software Design and Collections.

Challenging exercises are indicated with a * before their number.

Recursion

1. Consider the following method

```
public static void cheers(int n) {
    if ( n<=1)
        System.out.println("Hurrah");
    else {
        System.out.println("Hip ");
        cheers(n-1);
    }
}
```

What is the output of `cheers(3)`?

2. Modify the `cheers` method in the previous exercise so that it first prints “Hurrah” followed by $n-1$ “Hip”s.
3. a. Make a further modification of the `cheers` method to to print $n-1$ “Hip”s before and $n-1$ “Hip”s after the “Hurrah”.

b. Make another modification so that half of the $n-1$ “Hip”s occur before and half appear after the “Hurrah”. It should always print the same number of “Hip”s on both sides of “Hurrah”. Therefore, when n is even it may print n “Hip”s overall instead of $n-1$ "Hip"s.

4. A string that is the same when read forwards or backwards is called a *palindrome*. Consider the following code segment:

```
String word = "hello";
String word2 = "alla";
String word3 = "ogopogo";

System.out.println(word + ": " + isPalindrome(word));
System.out.println(word2 + ": " + isPalindrome(word2));
System.out.println(word3 + ": " + isPalindrome(word3));
```

The output of this code is:

```
hello: false
alla: true
ogopogo: true
```

Write a **recursive** static method `isPalindrome()` that takes a string as its parameter and returns a boolean that indicates whether the argument is a palindrome or not.

You may assume that the string contains only letters and that all are of lower case.

- *5. Examine the following pattern of asterisks and blanks:

```
*
* *
  *
* * * *
  *
  * *
    *
* * * * * * * *
  *
  * *
    *
  * * * *
    *
    * *
      *
```

Write a recursive function that can generate this pattern. Your method should be able to produce larger and smaller patterns with the same structure. [Hint: Use two parameters, one to indicate the indentation, and one for the number of stars.]

More on Collections

1. Suppose that a Dog class is defined as follows:

```
public class Dog {
    private String breed;
    private String name;
    private String gender;

    public Dog(String aBreed, String aName, String aGender)
    { ... }
    public String getBreed() { ... }
    public String getName() { ... }
    public String getGender() { ... }

    /* Two dogs are equal if they are of the same breed,
       and gender and have the same name.
    */
    public boolean equals(Object o) {...}
    ...
}
```

We want to define a class DogRegistry that contains a collection of all the dogs in Vancouver. Suppose the two operations that we will want to apply most frequently to this class are adding a dog to the registry, and returning a collection of all the registered dogs of a given breed. What would be the most efficient collection for the implementation of the DogRegistry? Complete the declaration of this collection and implement the methods addDog and getDogsOfBreed:

```
public class DogRegistry {

    // Declaration of the collection component

    ...
    public void addDog(Dog dog) {
    ...
    }

    public Set<Dog> getDogsOfBreed( String breed) {
    ...
    }
    ...
}
```

2. Provide an implementation for the following utility method named `removeValuesFromMap` that takes a `Map<K,V>` and an `item` of type `V` as its only parameters and that removes all the key-value pairs for which the value is equal to `item`. Assume that the `equals()` method has been overridden for objects of type `V`.

```
public static <K,V> void removeValuesFromMap( Map<K,V> data, V item )  
{
```

Stacks & Queues

1. Given the following stack `s`:

```
C ← top  
B  
A
```

Show how the stack will look after each of the following sequences of operations (each sequence starts with the original stack shown above):

- a. `s.pop(); s.pop(); s.push(D); s.pop(), s.push(E)`
- b. `s.pop(); s.pop(); s.pop(); s.pop();`

2. Write the code for the method

```
public static <E> void reverse( Stack<E> stack )
```

which reverses its argument stack. Your code can only use a `Queue` to reverse the given stack. No other variable of any type of other structure is allowed. You should assume that the class `LinkedList<E>` exists that implements the `Queue<E>` interface.

Streams

1. Write a static method (imagining that it is part of a utility class) that accepts two arguments: the name of a text file and a string. The method must print to standard output all lines in the given text file that contain the given string. (Hint: buffered streams have a method `readLine()` that returns the next line of a text file). Please note that on the final you will be provided with appropriate reference material for Java APIs including `Stream` classes.

Threads

1. After learning about threads, John decided to use them in a sorting program he was writing for another assignment. His idea is to create a thread that will sort an array of numbers at the same time that the main program is executing. He came up with the following program.

```
public class Sorter extends Thread {
    private int[] data = null;

    public Sorter( int[] data) {
        this.data = data;
    }

    public void run() {
        // Sort the data; This code is correct
        for (int i = 0; i < data.length - 1; i++)
            for (int j = 0; j < data.length - 1; j++)
                if (data[j] < data[i]) {
                    int tmp = data[i];
                    data[i] = data[j];
                    data[j] = tmp;
                }
    }

    public static void main( String args[] ) {
        int[] data = new int[10];

        // generate some random data to test
        for (int i = 0; i < data.length ; i++)
            data[i] = (int) (Math.random() * 100);

        Sorter x = new Sorter( data );
        x.start();

        for (int i = 0; i < data.length ; i++)
            System.out.println( data[i] );
    }
}
```

The program is acting weird. The numbers printed out are sometimes sorted, sometimes are partially sorted, and sometimes are not sorted at all. Find out what is wrong with this program and what you need to do to fix it. Note that since sometimes the program prints the numbers sorted, you can assume that the sort routine is working correctly.

2. Write a method named `joinAll()` that takes as a parameter an array of threads and returns after all the threads in the array have been terminated.

3. In this question we imagine that we are trying to model a network connection between two machines and that multiple threads are making connections between machines. In the simplified example below, we have only two machines and two threads – one thread for each machine. Each thread tries to repeatedly establish a connection from its machine to the other machine and send a message. To establish the connection, we call `connect()` on one machine and give it a reference to the other machine. The other machine must then `acknowledge()` the request before a message can be sent. The code below represents an attempt to model this problem. Note that the detail of how to send a message between the machines is omitted.

```
public class Machine {
    private Lock machineLock;
    private String name;

    public Machine( String name ) {
        this.name = name;
        machineLock = new ReentrantLock();
    }

    public void connect( Machine other ) {
        machineLock.lock();
        try {
            System.out.println( "Connecting " + name + " to " + other.name );
            other.acknowledge();
            System.out.println( "Connection established." );
        }
        finally {
            machineLock.unlock();
        }
    }

    public void acknowledge() {
        machineLock.lock();
        try {
            System.out.println( name + " acknowledged connection." );
        }
        finally {
            machineLock.unlock();
        }
    }
}
```

```

public class NetworkAdmin extends Thread {
    private Machine machine;
    private Machine other;

    public NetworkAdmin( Machine m1, Machine m2 ) {
        machine = m1;
        other = m2;
    }

    public void run() {
        while( true ) {
            machine.connect( other );
            //send message
            //...
            //drop connection
        }
    }
}

public class NetworkApp {
    public static void main( String[] args ) {
        Machine m1 = new Machine( "Machine 1" );
        Machine m2 = new Machine( "Machine 2" );

        NetworkAdmin na1 = new NetworkAdmin( m1, m2 );
        NetworkAdmin na2 = new NetworkAdmin( m2, m1 );

        na1.start();
        na2.start();
    }
}

```

When the application runs, the following is displayed on the console:

```

Connecting Machine 2 to Machine 1
Machine 1 acknowledged connection.
Connection established.
Connecting Machine 1 to Machine 2
Connecting Machine 2 to Machine 1

```

after which there is no further output on the console.

- a) What is the formal name given to the problem illustrated in this code?

- b) Explain why this problem has arisen but do not attempt to explain how to fix it. Your explanation must include a possible sequence of method calls (starting with `machine.connect(other);`) run by each thread (`na1` and `na2`) that gives rise to the *last two lines* of output above. You should provide your explanation in point form – one point for each method call. Your explanation must include the calls to `lock()` and `unlock()` and the effect that these calls have on the ability to execute the enclosed critical section(s) of code. Assume that no lock is held at the start of your sequence of method calls.