Software Development I
Univ.-Prof. Dr. Alois Ferscha
Examination, January 27, 2014

Institute for Pervasive Computing

Last name:	**SAMPLE SOLUTION**		Lecture hall:	
First name:			Seat:	
ID:		SKZ:	Points / Grade:	

- **Please fill in the header of your test** and make sure that you have your **student ID** („Keplerkarte") at hand
- You may use resources such as Java textbooks, lecture slides, assignments, and other written material
- It is not allowed to use any electronic devices (cell phones, PDAs, notebooks, iPads, calculators)
- It is not allowed to use extra paper
- **Please do not use pencils or red/green pens**
- Solutions have to be provided in the framed placeholders
- The score of each (part of an) example is an indication of the intended editing time (i.e.**, 90 points in total corresponds to 90 minutes editing time**).

GOOD LUCK!

1. Recursion (4+6=10 points)

What happens when you run the following code, that is, what (if anything) gets written as an output and what goes wrong? After answering those questions, fix the implementation of `hasSubstring` so that the code below does not crash and prints the correct output (which is: `true`, `false`, `true`, `true`, `false`).

```
public class Recursion {

1   private static final String TEXT = "This is the text to be searched";
2
3   private static boolean hasSubstring(String toFind, String findFrom) {
4
5       if (findFrom.startsWith(toFind)) {
6           return true;
7       }
8
9       return hasSubstring(toFind, findFrom.substring(1));
10  }
11
12  public static void main(String[] argv) {
13      System.out.println(hasSubstring("text to", TEXT));
14      System.out.println(hasSubstring("goo", TEXT));
15      System.out.println(hasSubstring("This", TEXT));
16      System.out.println(hasSubstring("searched", TEXT));
17      System.out.println(hasSubstring("the  text", TEXT));
    }
}
```

What is the output of the program?

```
// The code prints a single "true" after which it crashes because of a
// StringIndexOutOfBoundsException (the exact exception must not be named in the answer).
// 4 points
```

Fix (indicate at which line of the original code the fix should be put into):

```
        // insert after lines 4 (slightly different behavior) or 8:

        if (findFrom.length() == 0) {
            return false;
        }
        // 6 points
```

2. Arrays/Loops (4+2+4=10 points)

We are looking for a JAVA program that is printing prime numbers from 1 to any specified number. This program consists of two parts: The first part is taking input from a user to print prime numbers and the second part is the function `isPrime(int number)` which checks whether a number is prime or not.

The rule for prime number checking is as follows:

- **A number is called prime number if its not divisible by any number other than 1 or itself.**
- **The first prime numbers are:** 2, 3, 5, 7, 11, 13, 17, 19, 23, 29

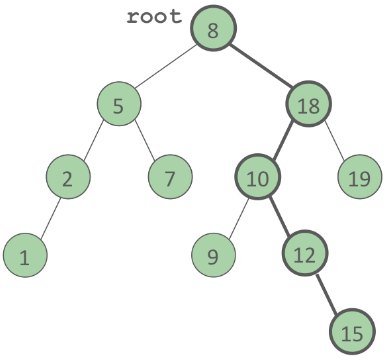
Complete the `main()` and `isPrime(int number)` methods.

```
public class PrintPrimeNumbers {

    public static void main(String args[]) {

        // User input till which prime number to check (and print results)
        System.out.println("Enter the number till which prime number to be printed: ");
        int limit = Input.readInt();

        // printing prime numbers from 1 to 'limit'; note: 1 is not a prime number!
        // note: make use of ‘isPrime()’
        for(int number = 2; number <= limit; number++) {
            //print prime numbers only
            if(isPrime(number)) {
                System.out.println(number);
            }

        }
    }

    // Prime number is not divisible by any number other than 1 and itself
    // @return true if number is prime
    public static boolean isPrime (int number) {

        for ( int i=2; i<number; i++ ) {

            if(number %i == 0){
                return false; //number is divisible so its not prime
            }

        }

        return true; //number is prime now
    }
}
```

3. Binary Trees (5+10=15 points)

Given a binary tree, compute the number of nodes along the longest path from the root node down to the farthest leaf node. The `maxDepth()` method should return 0 (zero) for an emptry tree, the 'root-to-leaf'-depth of the tree otherwise (i.e., the maximum number of elements from the root to a leave).

```
public class Node {
    int data;
    Node left;
    Node right;

    Node (int data) {
        left = null;
        right = null;
        this.data = data;
    }
}
```



maxDepth(root) = 5

```
public class BinTree {

    // Root node pointer ('null' for an empty tree)
    private Node root;

    /**
     * 'maxDepth' returns the 'root-to-leaf'-depth of the tree (max. number of elements from
     * root to a leave).
     * It uses a recursive helper that recurs down to find the max depth.
     */
    public int maxDepth() {
        return maxDepth(root);
    }

    private int maxDepth(Node node) {

        // maxDepth for an empty tree is zero. (=recursion anchor!)


        // correct recursion anchor: condition (3 points) + return with 0 (2 points)
        if (node==null) {
            return 0 ;
        }


        else {
            // recursive calls; use the larger result for further processing

            // two recursive calls with correct parameters (5 points)
            int lDepth = maxDepth (node.left);
            int rDepth = maxDepth (node.right);

            // return (1 point), use the larger value (3 points), add 1 (1 point)
            return (Math.max(lDepth, rDepth) + 1);

        }
    }
```

4. String Processing (7+3=10 points)

Complete the method `computeInitials()` that computes the initials from a full name and returns the result, e.g. `computeInitials("Andreas Riener")`would result in `"AR"`.

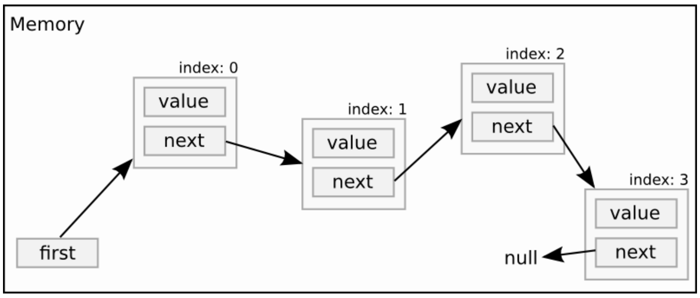
Notes

- The original string `'name'` must not be modified.
- **The result string contains all uppercase letters from the original string, no blanks between characters.**
- You can use all built-in methods from the classes `String/StringBuffer` and `Character`, in particular
  - `Character.isUpperCase(char c)`
    …returns `true` if `'c'` is an upper case character
  - `string.charAt(int index)`
    …returns the char `'value'` at the specified index.
  - `strbuf.append(c)`
    …appends the character `'c'` to the StringBuffer `strbuf`
  - `strbuf.toString()`
    …returns a string representation of the sequence of characters in `strbuf`
- **Be careful with the return value!**

```java
public class ComputeInitials {

    public static void main(String[] args) {

        String name = "Bruce M. Walker";

        System.out.println("Initials: " + computeInitials (name)); // output: "BMW"
    }

    public static String computeInitials (String name) {
        StringBuffer initials = new StringBuffer();
        int length = name.length();

        for (int i = 0; i < length; i++) {
            // check every character if its upper case; if so, append it to the result string

            // check for upper case  (4 points), append to initials (3 points)
            if (Character.isUpperCase(name.charAt(i))) {
                initials.append(name.charAt(i));
            }

        }
        // return the result (string of initials)

        // return (1 point), correct value (type String!) (2 points)
        return initials.toString();

    }
}
```

5. Singly Linked List (7+8+10=25 points)

On the right you see a figure describing a linked list and assume that you have also been provided with the `LinkedListItem` class that already implements the following variables and methods:



```
public class LinkedListItem {
    private String value;
    private LinkedListItem next;

    public LinkedListItem(String contents);    // The constructor!
    public void setNext(LinkedListItem next); // Set the next item in chain
    public LinkedListItem getNext();           // Get the next item in chain
    public String getContents();               // Get the contents of the item
}
```

Fill in the missing code in the partial implementation below to make the source code complete. Note that the list index starts at 0, i.e. the first element in the list has index 0.

```
public class LinkedList {

    LinkedListItem first;

    public LinkedList() {
    }

    /* method returns 'null' if target not found or invalid index. */
    private LinkedListItem getItemAt(int idx) {
        LinkedListItem target = first;
        if (idx < 0)
            return null;
        // FILL IN HERE: searching for the correct item in the list

        // while loop + condition (5 points), next reference (2 points)
        while (idx > 0 && target != null) {
            target = target.getNext();
            idx--;
        }

        return target;
    }

    public String get(int idx) {
        LinkedListItem item = getItemAt(idx);
        if (item == null) {
            return null;
        } else {
            return item.getContents();
        }
    }
    /* method adds a new node with value 'text' at position 'idx' to the list. */
    public boolean addTo(int idx, String text) {
        if (idx < 0) return false;

        /* special case: add as first element */
        if (idx == 0) {
            // FILL IN HERE: adding a new item to the front of list

              // instantiate and parametrize new node (3 points)
              LinkedListItem newFirst = new LinkedListItem(text);
              newFirst.setNext(first); // add as first element (3 points)
              first = newFirst;        // change ‘first’ reference (2 points)

            return true;
        }

        /* get the previous item */
        LinkedListItem prev = getItemAt(idx-1);
        if (prev == null) {
            /* can't add as idx, we don't have that many items */
            return false;
        }
        // FILL IN HERE: adding a new item after the previous item

        // instantiate and parametrize new node (3 points)
        LinkedListItem newItem = new LinkedListItem(text);
        newItem.setNext(prev.getNext()); // add new item at correct place (4 points)
        prev.setNext(newItem);           // set ‘next’ pointer of prev node (3 points)

        return true;
    }
}
```

6. Objects/Classes/Methods (2+4+3+4+4+3=20 points)

The class `Point` models two-dimensional points with x and y coordinates. Complete the implementation of the class as described in the following.

1. **Instance variables**

   Define two private instance variables for the x and y coordinates (both of type `int`)

2. **Constructors**

   a. Add a standard (“no argument”) constructor that constructs a point at coordinates (0,0).

   b. Add another constructor that constructs a point with given x and y coordinates

3. **Instance methods**
   a. Implement a method called `distance` that takes two `int` values x and y as input and returns the distance (type `double`) from the actual point to another point at the given (x, y) coordinates.
   b. Implement an **overloaded** method `distance` that takes another point as input and returns the distance (type `double`) from the actual point to the given other point (instance of `Point`).
4. **Class methods**
   a. Add a class method `distance` that takes two points (each instances of `Point`) as input, and returns the distance between the two points (type `double`).

Notes

- Your solutions should be as efficient as possible, e.g. **re-use of code**
- Be careful with **visibility** of variables and methods

```java
// class Point models a 2D point with x and y coordinates.
// Aim: Implement different variants of ‘distance’ to calculate the distance between 2 points
public class Point {
    // 1. Instance variables
    private int x; // (2*1 points)
    private int y;


    // 2. Constructors

    public Point() { // (2*2 points)
        this.x = 0;
        this.y = 0;
    }

    public Point (int x, int y) {
        this.x = x;
        this.y = y;
    }


    // 3. Instance methods ‘distance’
    // Note: The distance between two points calculates to d=Math.sqrt(deltaX^2+deltaY^2)
    // a. input: two coordinates x, y (int), returns distance (as double)

    public double distance (int x, int y) { // (4 points)
        int xDiff = this.x - x;
        int yDiff = this.y - y;

        return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
    }
    // b. input: a point (type Point), returns distance (as double)
    public double distance (Point p) { // (4 points)

        return distance (p.x, p.y);

        // less efficient alternative
        // int xDiff = this.x - p.x;
        // int yDiff = this.y - p.y;
        // return Math.sqrt(xDiff*xDiff + yDiff*yDiff);
    }

    // 3. Class method. Input: two points (type Point), returns distance (as double)
    public static double distance (Point p1, Point p2) { // (3 points)

        return p1.distance(p2);
    }
}
```

Complete finally the test program outlined below.

```
// class Point models a 2D point with x and y coordinates.
// Aim: Implement different variants of ‘distance’ to calculate the distance between 2 points
// module to test correct behavior of class ‘Point’
...
    public static void main(String[] args) {
        Point p1 = new Point (3, 0);
        Point p2 = new Point (0, 4);

        // check instance method ‘distance’: distance between p1 and a point defined by (0,4)

        System.out.println(  p1.distance(0, 4)  // (3*1 points)          );

        // check instance method ‘distance’ : distance between p1 and p2

        System.out.println(  p1.distance(p2)                               );

        // check class method ‘distance’: STATIC calculation of the distance between p1 and p2

        System.out.println( distance(p1,p2)                                );

    }
}
```