The interface for the String class from the Java documentation is attached at the end of this paper. Please detach and use when needed through all parts of this paper.

Part A: Short Answer Questions (60 marks). Answer all questions.

1. What is a **keyword**? Give two examples, using Java as the programming language. (2 marks)

2. What will be the **output** for the following program segment? (2 marks)

```
System.out.println(10%5);
System.out.println(10/5);
System.out.println(10<10);
System.out.println(10==10);
```

3. What is the difference between a **constructor** and a **class method**? (2 marks)

4. Create a class called DoSomething, that has two integer fields, int1 and int2. Initialise these fields in a constructor. (3 marks)

5. Rewrite the following "if" construct as a **switch/case statement**: (2 marks)

```
char code;

if (code =='A') {
   System.out.println("Excellent");
} else if (code =='B' || code == 'C') {
  System.out.println("Good");
} else if (code == 'D') {
   System.out.println("Poor");
}
```

6. Convert the following code segment into an equivalent **for loop** (2 marks)

```
public void display()
{
  int i  = 10;
  while (i !=20)
  {
    System.out.println(i + " ");
    i++;
  }constructor


}
```

7. What will be the **output** of the following program segment? (2 marks)

```
int a = 10;
int b = 30;
int c = 40;
a = b-- + c - a % 2;
System.out.println("a"+a);
System.out.println("b"+b);
```

8. What is the difference between a **field** and a **local variable**? (2 marks)

9. What is a **Wrapper class**? Give <u>two examples</u> of a Wrapper class (2 marks)

10. What is the purpose of a **break** statement? (2 marks)

11. <u>For each</u> of the Java method signatures below, answer the following questions:

    a. What is the method name?
    b. Does the method return a value? If yes, what is the type of the return value?
    c. How many parameters does the method have?
    d. Write a sample Java method call for the method signature; that is: how would you call the method in your code?

public void changeColour(String colour)	(2 marks)
public int calculateLargest(int A, int B)	(2 marks)
public boolean hasEaten()	(2 marks)

12. Write a constructor for the Date class given the following sample object creation code (2 marks):

```
new Date("March", 30, 1932);
```

Write another constructor that takes no parameters, for the same class (2 marks).

13. Explain the difference between **accessor** and **mutator** methods (2 marks).

14. What is the difference between the following statements? (1 mark)

```
System.out.println("Hello");
```

and

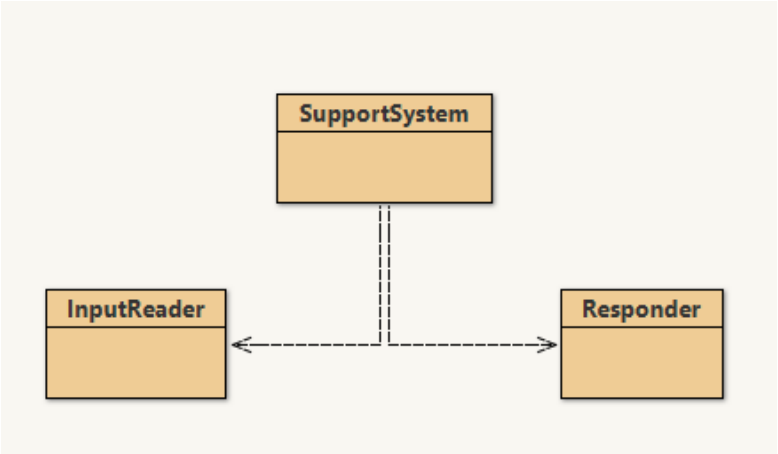
```
System.out.print("Hello");
```

15. Why are **method stubs** used when coding classes and projects? (2 marks)

16. Create a **truth table** to help interpret the possible results of the following boolean expression. Assume a, b and c are boolean variables. (2 marks)

```
if ((a & b) || c) {
   // do something
}
```

17. Given the following class diagram, explain the relationship between the three classes (2 marks):



18. Write a declaration for a new **ArrayList** of Person objects, called students. Initialise the new ArrayList. Note: you may leave it empty. (2 marks)

19. What sort of **loop** is most commonly used to iterate through collections? (1 mark)
    If the Person class in Q18 has a method called "printName" that takes no parameters, and returns a String, **write a loop** that will print out all the students' names, one per line. (2 marks).

20. What are the three main **control structures** in programming? (1 mark)

21. Name two ways of **producing output** for the user. Use examples to illustrate your answer. (2 marks).

22. When we create a method that reads from or prints to a file, we need to include the words "**throws Exception**" to the header. Why do we need to include these keywords? (2 marks)

23. Describe the meaning of **cohesion**, in terms of writing classes. Give an example of why cohesion is important (3 marks).

24. Briefly describe the concept of **information hiding.** (2 marks)

25. What is the difference between a method which is static and one which is not static? (1 mark)

Part B: Java Programming Problems. Choose 2 of the following problems. Each solution is worth 20 marks. You are expected to use comments in your code. Marks will be deducted for incorrect brackets, syntax etc, so be careful as possible (40 marks).

1. Write a program to enter any ten words from the keyboard, arrange them in alphabetical order (lexicographically) and print them in order in the terminal window.

2. Write a program in Java to accept a line of text from the user and create a new word formed out of the first letter of each word.
   For example, the input line "Mangoes are delivered after midnight" would produce "Madam".

3. Write overloaded methods in Java to handle the following situation – a Shape class is created that has (at least one) method called calcArea(). Users should be able to use this method in the following way:

```
// assuming r is a Shape representing a rectangle, s is a Shape representing a square,
// and c is a Shape representing a circle:
System.out.println("The area of the rectangle is " + r.calcArea(6, 10);
System.out.println("The area of the square is " + s.calcArea(25);
System.out.println("The area of the circle is " + c.calcArea(4.5);
```

4. Write a program that inputs 10 numbers, and prints out the second largest number with a suitable message.

5. Write a program in java to print the following pattern.

```
1
1    5
1    5    10
1    5    10    15    20
```

(and so on) until the final number on a line is 50.

Part C: Optional extensions (max of 10 extra points, maximum overall of 100 marks)

1. Explain what the Clean and Build command does in the Netbeans environment (5 marks). Make sure you explain the result of this command as well as the mechanics of what it does.

2. What is the purpose of a code template in the Netbeans editor? What are their advantages and disadvantages? (5 marks).

END OF EXAM

--------------------------------------page deliberately left blank ---

String documentation:

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(**String** anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(**String** str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(**String** str) Concatenates the specified string to the end of this string.
boolean	**contains**(**CharSequence** s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(**CharSequence** cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(**StringBuffer** sb) Compares this string to the specified StringBuffer.
static **String**	**copyValueOf**(char[] data) Equivalent to **valueOf(char[])**.
static **String**	**copyValueOf**(char[] data, int offset, int count) Equivalent to **valueOf(char[], int, int)**.
boolean	**endsWith**(**String** suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(**Object** anObject) Compares this string to the specified object.

boolean	**equalsIgnoreCase**(**String** anotherString) Compares this String to another String, ignoring case considerations.
static **String**	**format**(**Locale** l, **String** format, **Object**... args) Returns a formatted string using the specified locale, format string, and arguments.
static **String**	**format**(**String** format, **Object**... args) Returns a formatted string using the specified format string and arguments.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
byte[]	**getBytes**(**Charset** charset) Encodes this String into a sequence of bytes using the given **charset**, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the **getBytes()** method, which uses the platform's default charset.
byte[]	**getBytes**(**String** charsetName) Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hash code for this string.

int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(**String** str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(**String** str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.

boolean	**isEmpty**() Returns true if, and only if, **length()** is 0.
static **String**	**join**(**CharSequence** delimiter, **CharSequence**... elements) Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
static **String**	**join**(**CharSequence** delimiter, **Iterable**<? extends **CharSequence**> elements) Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(**String** str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(**String** str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(**String** regex) Tells whether or not this string matches the given **regular expression**.

int	**offsetByCodePoints**(int index, int codePointOffset) Returns the index within this String that is offset from the given index by codePointOffset code points.
boolean	**regionMatches**(boolean ignoreCase, int toffset, **String** other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, **String** other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**replace**(**CharSequence** target, **CharSequence** replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.

String	**replaceAll**(**String** regex, **String** replacement) Replaces each substring of this string that matches the given **regular expression** with the given replacement.
String	**replaceFirst**(**String** regex, **String** replacement) Replaces the first substring of this string that matches the given **regular expression** with the given replacement.
String[]	**split**(**String** regex) Splits this string around matches of the given **regular expression**.
String[]	**split**(**String** regex, int limit) Splits this string around matches of the given **regular expression**.
boolean	**startsWith**(**String** prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(**String** prefix, int toffset) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
CharSequence	**subSequence**(int beginIndex, int endIndex) Returns a character sequence that is a subsequence of this sequence.
String	**substring**(int beginIndex) Returns a string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toLowerCase**(**Locale** locale) Converts all of the characters in this String to lower case using the rules of the given Locale.
String	**toString**() This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.
String	**toUpperCase**(**Locale** locale) Converts all of the characters in this String to upper case using the rules of the given Locale.
String	**trim**() Returns a string whose value is this string, with any leading and trailing whitespace removed.

static **String**	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static **String**	**valueOf**(char c) Returns the string representation of the char argument.
static **String**	**valueOf**(char[] data) Returns the string representation of the char array argument.
static **String**	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static **String**	**valueOf**(double d) Returns the string representation of the double argument.
static **String**	**valueOf**(float f) Returns the string representation of the float argument.
static **String**	**valueOf**(int i) Returns the string representation of the int argument.
static **String**	**valueOf**(long l) Returns the string representation of the long argument.
static **String**	**valueOf**(**Object** obj) Returns the string representation of the Object argument.