Java By Abstraction - Test-B (Chapters 1-6)

Last Name	
First Name	

Do not write below this line

			A (40%)	
B (60%)				
			TOTAL	

String Methods (invoke on a string s)	`char` **`charAt(int p)`** Returns the character at position# p in s.
`boolean` **`equals(String t)`** Returns *true* if s and t have equal contents.	`int` **`compareTo(String t)`** Returns a negative number if s<t, zero if s=t, and a positive number if s>t.

`int` **`indexOf(String t, int f)`**
Looks for the string t within s, starting at position# f in s. Returns the position in s where the match was found. Returns -1 if no match was found.
	`Integer.`**`parseInt`**`(s)`
	`Double.`**`parseDouble`**`(s)`
	Static methods to convert a string **s** that contains a number to a primitive type.

`int` **`indexOf(String t)`** Looks for the string t within s (as above), starting at the beginning of s.	`String` **`trim()`** Returns the same content as s but with any leading/trailing white-space removed.
`String` **`substring(int f,int t)`** Returns all characters in s with position numbers ≥ f and < t.	**Static methods in Math**
`String` **`substring(int f)`** Returns a substring of s that begins at f and extends to the end of s.	`double` **`abs(double x)`** Returns the absolute value of x.
`String` **`replace(char x,char y)`** Returns a string with all occurrences of character x in s replaced by y.	`double` **`pow(double x, double y)`** Returns x raised to y.
`String` **`toUpper/LowerCase()`** Returns a string of all characters in s converted to upper / lower case.	`double` **`rint`**`(double a)` Returns the closest `double` value to a that is equal to a mathematical integer.

GROUP - A *<10 questions x 4 points each = 40 points >*

For each question, write in the box the output of the shown fragment. If you believe the fragment will not produce output due to errors, write the error type and a brief explanation. You can assume all needed classes are properly imported.

A.1
```
int k = 16412;
int m = k % 10;
k = k / 10;
k = k % 1000;
IO.println(m);
IO.println(k);
```

A.2
```
String s = "York University";
int p = s.indexOf("n");
int q = s.substring(3, 9).indexOf("n");
IO.println(p);
IO.println(q);
```

A.3
```
Stock s = new Stock("NT");
Stock u = new Stock();
Stock t = s;
u.setSymbol("NT");
IO.println(t.getSymbol());

s.setSymbol("ATY");
IO.println(t.equals(u));
```

A.4

```
int i1, i2, i3;
double r1, r2;
i1 = 5;
r1 = 3.0;
r2 = 2 * r1;
i2 = 2 * i1;
i3 = i2 / r2;
IO.println(r2);
IO.println(i3);
```

A.5 The `BankAccount` class has a constructor that takes two parameters: the name of the account (a `String`) and its initial balance (a `double`). The class has a `double` accessor method `getBalance()` that returns the balance of the account on which it was invoked.

```
BankAccount a1 = new BankAccount("Mary", 1000);
BankAccount a2 = new BankAccount("Mary", 1000);
BankAccount a3;
a3 = a1;
int m = 0;
if (a2 == a3)
{  m = 55;
}
int k = 0;
if (a2 == a1)
{  k = -1
} else if (a3.getBalance() == a2.getBalance())
{  k = -22
} else
{  k = -333
}
IO.println(m);
IO.println(k);
```

A.6
```
int x = 5763;
int y = 0;
int k;
for (k = 0; x > 0; k++)
{  y = y + x % 10;
   x = x / 10;
}
IO.println(k);
IO.println(y);
```

A.7
```
int a = 10;
int b = 20;
int c = 30;
boolean m = a + 10 == c;
IO.println(m);

if (a > b || b + 10 > c)
   IO.println("case 1");
else if (a < b && b + 10 < c)
   IO.println("case 2");
else if ( !(a + 20 > c) || a + 10 < b)
   IO.println("case 3");
else
   IO.println("case 4");
```

A.8
```
String s = "abcdefgh";
int k = s.length() - 6;
IO.println(s.substring(k, s.length() - 1));
boolean b = s.substring(1,2) == "b";
IO.println(b);
```

A.9
```
String s1 = "100";
String s2 = "20";
IO.println(s1 + s2 +  30);
IO.println(9 / 2 + 30 + s1 + s2);
```

A.10
```
Stock stk1 = new Stock("RY");
Stock stk2 = new Stock("BMO");
Stock stk3 = stk2;
stk2 = null;
boolean b1 = stk2 == stk3;
IO.println(b1);

stk3.setSymbol("RY");
boolean b2 = stk3 == stk1;
IO.println(b2);
```

GROUP - B *<60 points >*

B.1 <20 points>

Consider the following (partial) API of two classes:

Department Class

Constructor Summary	
`Department(String name, int budget)` Constructs a `Department` object. **Parameters:** name - name of the department budget - budget of this department	
Method Summary	
`void`	`assign(Employee who)` Add an employee to `this` department **Parameter:** who - the employee to be assigned to `this` department
`int`	`getHeadCount()` **Returns:** the number of employees in `this` department
`int`	`getBudget()` **Returns:** the budget of `this` department
`void`	`changeBudget(int delta)` Increase or decrease the budget of `this` department by the passed amount. Delta is the increment or decrement, *not* the new budget. **Parameter:** delta - change the budget by adding this amount to it *(to reduce budget, provide a negative amount).*

Employee Class

Constructor Summary
`Employee(String name,int rank)` Constructs an `Employee` object. **Parameters:** name - name of the employee rank - the rank (level) of the employee

Develop the Java application **`App`** whose main method performs the following tasks, in the order shown:

1. Create a department called "R&D" with budget 2,000,000.

2. Create an employee John whose rank is 3.

3. Create an employee Debbie whose rank is 2.

4. Assign both John and Debbie to the R&D department

5. Determine the head count of the R&D department by using a method, and store it in some variable `count`.

6. If `count` is greater than 10, increase the department budget by 5%, otherwise reduce it by 2%.

Note that it is OK to use the above magic numbers –no need to store them in finals.

Write your app on the next page.

```
import type.lang.*;

public class App
{  public static void main(String[] args)
   {
```

GROUP - B

B.2 <20 points>

Write the program App that starts by prompting for and reading a string from the user. If the length of the entered string is equal to or greater than 20, the program must terminate with the error message "`String too long!`". Otherwise, the program outputs the string after padding it with '+' characters at its two ends, so that the total length of the output is 20 and the entered string is at the centre of the output. If the number of '+' characters to be added is odd, you can put the extra '+' on either side. Three samples are shown.

```
Enter a string: Toronto
+++++++Toronto++++++
```

```
Enter a string: Computer Science Dept.
String too long!
```

```
Enter a string: 1234567890
+++++1234567890+++++
```

```
import type.lang.*;
public class App
{  public static void main(String[] args)
   {  final int WIDTH = 20;
```

GROUP - B

B.3 <20 points>

Write the program App that plays a game with the user as follows: it prompts the user to enter a guess for the role of the dice. It then simulates throwing one die by generating a random number (an integer between 1 and 6, inclusive) and displaying it on the screen. If the user's guess was correct, the user gets $2; i.e. the program adds $2 to the user's balance, otherwise, the user loses $2, and the new balance is displayed on the screen. The game continues indefinitely until the user enters an invalid guess (less than 1 or more than 6) or runs out of money. The user starts off with $10.

Here is a partial API of the `Random` class whose services enable you to simulate the throwing of a dice:

Random Class in the `java.util` package

Constructor Summary	
`Random()` Creates a new random number generator.	
Method Summary	
`int`	`nextInt(int n)` Returns a random number uniformly distributed between 0 (inclusive) and the specified parameter `n` (exclusive); i.e. the return is greater or equal to 0 and less than `n`.

Write the program on the next page.

```
import type.lang.*;

public class App
{  public static void main(String[] args)
   {
```