Recitation 3

Overview

- Functions
  - Function signature
  - Function params
  - void, return type
  - Error control
  - Overloading
- Overflow

Question 1 – Substrings

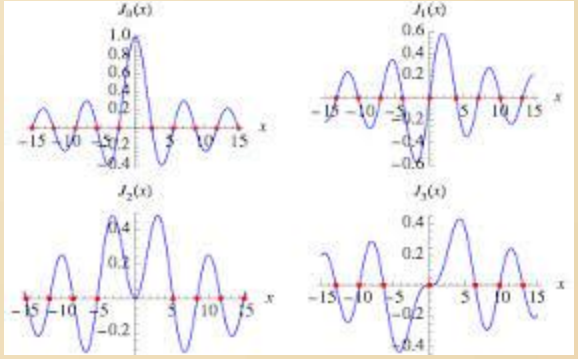
- A substring of a string, is any string which is wholly contained in the original string.
- Examples:
  - "kawa" is a substring of "kawa banga"
  - "banga" is a substring of "kawa banga"
  - "wa ba" is a substring of "kawa banga"
  - "hi" is not a substring of "kawa banga"
  - "kawabanga" is not a substring of "kawa banga"
- Design a program which does the following:
  - Receives two non-empty strings from the users.
  - Checks if the first string is a substring of the second.
  - Let the user know.

Question 1 - Solution

```
public class IsSubstring {
	public static void main(String[] args){
		String firstString = args[0];
		String secondString = args[1];
		boolean isSub = false;
		int lastIndex = secondString.length() – firstString.length() + 1;
		for (int i = 0; i < lastIndex && !isSub ; i++){
			if (firstString.charAt(0) == secondString.charAt(i)){
				isSub = true;
				for (int j = 0; j < firstString.length() && isSub ; j++){
				    isSub = (firstString.charAt(j) == secondString.charAt(i + j));
				}
			}
		}
		System.out.println(isSub);
	}
}
```

Recitation 3

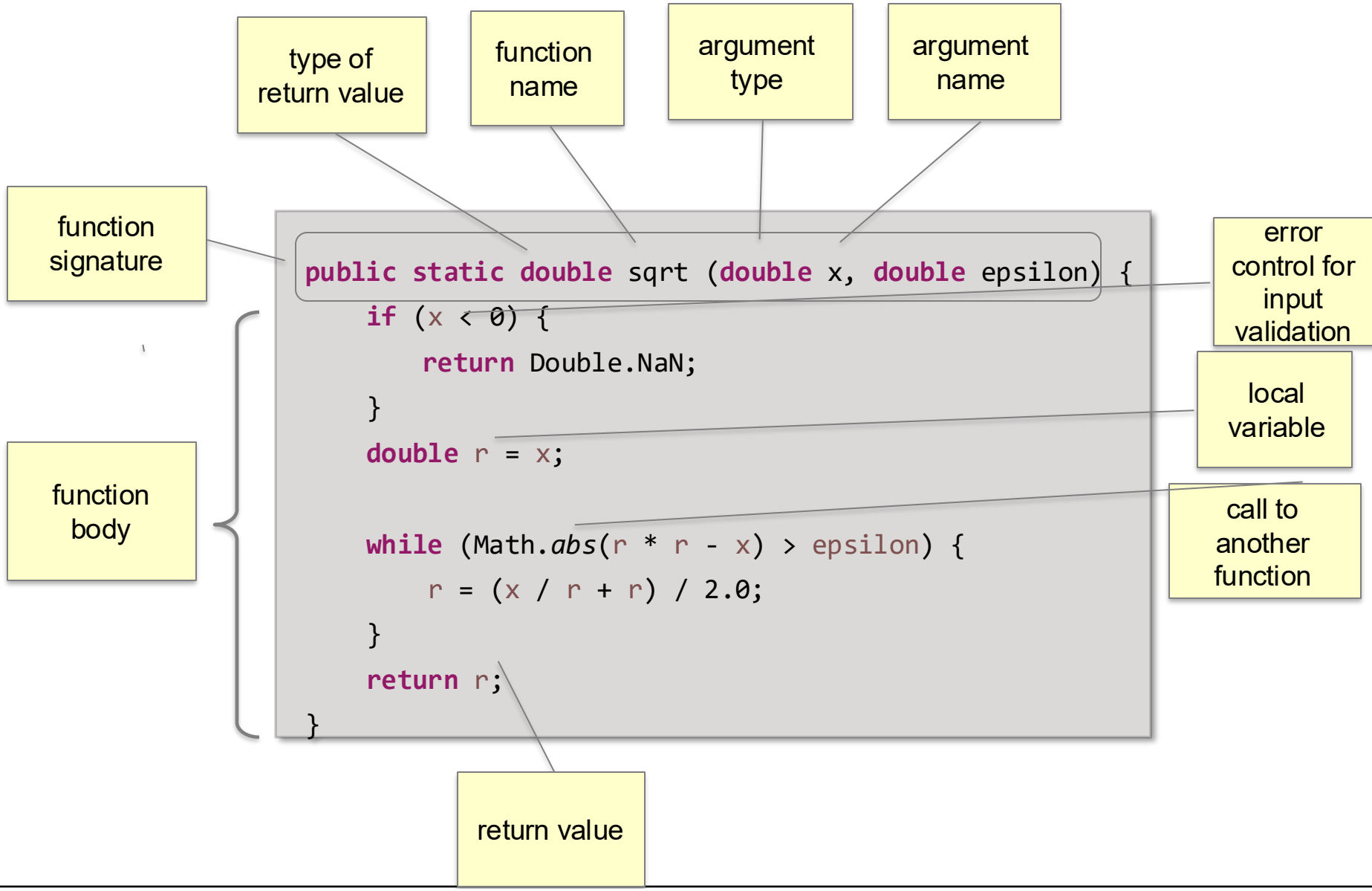
Functions



Functions

- Functions are independent part of a code which do a certain operation.
- Functions may have input and may have output.
- Functions are essential tool for developers.
- When I need to build a helper function ?
- DRY Method -> Don't Repeat Yourself
- If you have a code which has certain input, get certain output and don't influence the rest of your code.

Function anatomy



Determine the Outputs

```
public static void main(String[] args){
        int x = 1;
        System.out.println(x);//1
        add5(x);
        System.out.println(x);//1
}

public static void add5(int y){
        y = y + 5;
}
```

if the type of the return value is void means no value is returned

Determine the Outputs

```
public static void main(String[] args){
        int x = 1;
        System.out.println(x);//1
        add5(x);
        System.out.println(x);//1
}

public static int add5(int y){
        y = y + 5;
        return y;
}
```

return type must match the type mentioned in the signature. if not must me matched

Determine the Outputs

```
public static void main(String[] args){
        int x = 1;
        System.out.println(x);//1
        x = add5(x);
        System.out.println(x);//6
}

public static int add5(int y){
        y = y + 5;
        return y;
}
```

Question 1, Expansion 1 – Substrings

- A substring of a string, is any string which is wholly contained in the original string.
- Examples:
  - "kawa" is a substring of "kawa banga"
  - "banga" is a substring of "kawa banga"
  - "wa ba" is a substring of "kawa banga"
  - "hi" is not a substring of "kawa banga"
  - "kawabanga" is not a substring of "kawa banga"
- Design a **function** which:
  - Receives two non-empty strings from the users.
  - Checks if the first string is a substring of the second.

Question 1, Expansion 1 - Solution (Converted into function)

```
public static boolean isSubstring(String smaller, String bigger){
       boolean isSub = false;
       int lastIndex = bigger.length() – smaller.length() + 1;
       for (int i = 0; i < lastIndex && !isSub ; i++){
            if (smaller.charAt(0) == bigger.charAt(i)){
                isSub = true;
                for (int j = 0; j < smaller.length() && isSub ; j++){
                    isSub = (smaller.charAt(j) == bigger.charAt(i + j));
                }
            }
       }
       return isSub;
}
```

Question 1, Expansion 2 - Strong password

- To keep our information online secured, some applications require that we define a password for our account.
- To prevent hacking into our account, (and give the hackers a hard time) we need to choose strong passwords.
- In most applications a strong password must be at least n characters long, and contains at least 1 lowercase letter, 1 uppercase letter, and 1 digit.
- Write a function that receives a password (String) and returns true if it's a strong password or false if not under the following rules:
  - The password must be 6 character long.
  - The password must have at least 1 uppercase letter.
  - The password must have at least 1 lowercase letter.
  - The password must have at least 1 digit.

Question 1, Expansion 2 - Strong password

```
//Receives a String pass and returns if its a strong password
public static boolean isStongPassword(String pass) {

    boolean len = pass.length() >= 6;
    boolean hasUpper = false;
    boolean hasLower = false;
    boolean hasDigit = false;

    if (!len){
            return false;
    }
    for (int i = 0; i < pass.length(); i++){
            if (pass.charAt(i) >= 'a' && pass.charAt(i) <= 'z') {
                    hasLower = true;
            } else if (pass.charAt(i) >= 'A' && pass.charAt(i) <= 'Z') {
                    hasUpper = true;
            } else if (pass.charAt(i) >= '0' && pass.charAt(i) <= '9') {
                    hasDigit = true;
            }
    }
    return (capital && small && digit);
}

// discussion why the last line is not:  return (capital && small && digit && len); ?
```

Question 1, Expansion 2 - Strong password (another version, continue)

```java
//Receives a char ch and returns true if its a valid digit
public static boolean isDigit(char ch){
	return ch >= '0' && ch <= '9';
}

//Receives a char ch and returns true if its a valid upper case letter
public static boolean isUpper(char ch){
	return ch >= 'A' && ch <= 'Z';
}

//Receives a char ch and returns true if its a valid lower case letter
public static boolean isLower(char ch){
	return ch >= 'a' && ch <= 'z';
}
```

Question 1, Expansion 2 - Strong password (another version)

```
//Receives a String pass and returns if its a strong password
   public static boolean isStongPassword(String pass){

       boolean len = pass.length() >= 6;
       boolean hasUpper = false;
       boolean hasLower = false;
       boolean hasDigit = false;

       if (!len){
                 return false;
       }
      for (int i = 0; i < pass.length(); i++){
                 char current = pass.charAt(i);
                 if (isLower(current)) {
                           hasLower = true;
                 } else if (isUpper(current)) {
                           hasUpper = true;
                 } else if (isDigit(current)) {
                           hasDigit = true;
                 }
       }
       return (capital && small && digit);
   }

// discussion why the last line is not:  return (capital && small && digit && len); ?
```

Question 1, Expansion 2 - Strong password – testing solution

```
public class StrongPassword {
    public static void main (String[] args) {
        String password = args[0];
        boolean valid = isValid(password);

        if (valid) {
            System.out.println("The password is strong");
        } else {
            System.out.println("The password is not strong");
        }
    }

    public static boolean isStongPassword(String pass) {
        . . .
    }
}
```

```
% java StrongPassword yael

    the password is not valid

% java StrongPassword a3eT65e

    the password is valid
```

Question 1, Expansion 3 - Valid password

- Let's expand. Some applications go one step above it, and doesn't allow the password to contain the first name of the user.
- Write a function that receives a password (String), and a name (String of only letters) and returns true if it's a valid password based on the following conditions:
  - The password must be 6 character long.
  - The password must have at least 1 uppercase letter.
  - The password must have at least 1 lowercase letter.
  - The password must have at least 1 digit.
  - The password must not contain the name of the user.
    - You may assume that the name appears in lowercase inside the password.

Question 1, Expansion 3 - Solution

```
//Receives a String pass and returns if its a strong password
public static boolean isStongPassword(String pass, String name){
      boolean len = pass.length() >= 6;
      boolean capital = false;
      boolean small = false;
      boolean digit = false;

      if (!len || isSubstring(pass, name)){
         return false;
      }

      for (int i = 0; i < pass.length(); i++){
                char current = pass.charAt(i);
                if (isLower(current)) {
                          hasLower = true;
                }else if (isUpper(current)) {
                          hasUpper = true;
                }else if (isDigit(current)) {
                          hasDigit = true;
                }
    }
    return (capital && small && digit);
}
```

Question 1, Expansion 3 – testing solution

```
public class StrongPassword {
    public static void main (String[] args) {
        String password = args[0];
        String name = args[1];
        boolean valid = isValid(password, name);

        if (valid) {
            System.out.println("the password is valid");
        } else {
            System.out.println("the password is not valid");
        }
    }

    public static boolean isStrongPassword(String pass) {
        . . .
    }
}
```

```
% java ValidPassword yael dan
    the password is not valid
% java ValidPassword yaelL123 yael
    the password is not valid
% java ValidPassword a3eT65e yael
    the password is valid
```

Question 2 – Factorial

- One of the few functions which aren't supported by Math library is factorial.
- Factorial is a function which receives a nonnegative number (n) and return the multiplication of every positive number up to that point, the function is denoted **n!**:
  - 2! = 1 * 2
  - 5! = 1 * 2 * 3 * 4 * 5
  - 0! = 1
  - n! = 1 * 2 * …. * (n-1) * n
- The factorial function is used in permutations, combinations, and various other mathematical and statistical computations.
- Design a **function** which calculates the operation.

Question 2 – Factorial

```
public static int factorial(int n) {
        int ans = 1;
        for (int i = 1; i <= n; i++) {
                ans *= i;
        }
        return ans;
}
```

Question 2 – Factorial

- Let's test our function

```
public static void main(String [] args) {
        System.out.println(factorial(1));
        System.out.println(factorial(0));
        System.out.println(factorial(5));
        System.out.println(factorial(-1));
        System.out.println(factorial(20));
}
```

Expected output:

```
1
1
120
error of sort
2432902008176640000
```

Question 2 – Factorial

- Let's test our function

```
public static void main(String [] args) {
        System.out.println(factorial(1));
        System.out.println(factorial(0));
        System.out.println(factorial(5));
        System.out.println(factorial(-1));
        System.out.println(factorial(20));
}
```

Expected output:

1
1
120
error of sort
2432902008176640000

Actual output:

1
1
120
1
-2102132736

Question 2, Expansion 1 – Factorial, Error Control

- Let's focus first on the first error, and fix it,
- When you get an input which should be invalid, or rather have a value which indicates that something happen/didn't happen you want to return a value which will tell you what went wrong inside the function. We will want to return a value which represent an error of some sort.
- When the function returns a String value that is easy. However, what do I do when my function returns an int? or other type?
- You return a value which the function cannot return regularly. We already saw an example for that in the method str.indexOf(char ch).
- if the char 'ch' wasn't found, we will just return -1. Since in java there is no -1 index in the string.
- Example: "hello".indexOf('A');
  - Will result -1.

Actual output:

1
1
120
1
-2102132736

Question 2, Expansion 1 – Factorial, Error Control

- That for every non-negative integer n, the claim: n! <= (n+1)!, holds.
- Therefore, 0! is the smallest value which is 1 since 0, can have other purposes (next recitation) we will use -1.
- Why we rather use -1 over 3 ? Its easier to notice that there is something wrong.

```
public static int factorial(int n) {
        if (n < 0){
                return -1;
        }
        int ans = 1;
        for (int i = 1; i <= n; i++) {
                ans *= i;
        }
        return ans;
}
```

error control for input validation

Question 2, Expansion 1 – Factorial, Error Control

- Let's test our function after the change

```
public static void main(String [] args) {
        System.out.println(factorial(1));
        System.out.println(factorial(0));
        System.out.println(factorial(5));
        System.out.println(factorial(-1));
        System.out.println(factorial(20));
}
```

Expected output:

1
1
120
error of sort
2432902008176640000

Actual output:

1
1
120
-1 // changed
-2102132736

Question 2, Expansion 2 - What happened?

- Let's move to the other problem.
- First, let's understand why the value not make sense?
  - 20! = 1 * 2 * 3 * … * 20
  - As we recall, in factorial we only **multiply positive** numbers with other **positive** numbers, therefore, the result should be always be **positive**, yet the number -2102132736 is a **negative number.**
  - So, what happened here???

overflow

Question 2, Expansion 2 - Overflow

- What is overflow?
- As we know, each number representing whole number has a max value, and a min value.
- We both stored the value and returned it as int. If you recall **int** type has **max value** of 2,147,483,647, the value of 20! (2,432,902,008,176,640,000), exceeds it. Therefore, it moved to the **minimum** value of int.

Let's look at byte, to understand it with more manageable numbers.
Reminder: The min value of byte is -128, the max value of byte is 127.
If I was to write the following code:

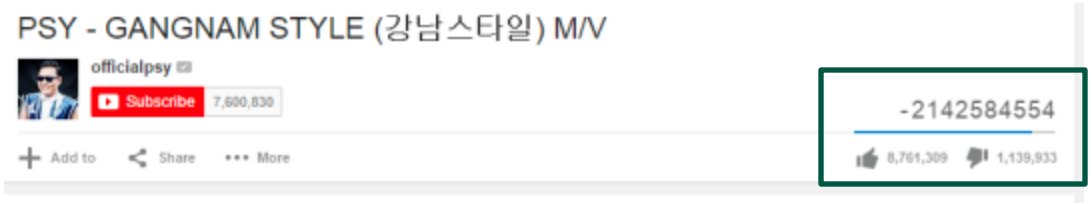
```
public static void main(String [] args) {
        byte b = 127;
        b += 1;
        System.out.println(b);
}
```

It will cycle back to the min value of byte, which is -128, hence b will be -128.

You must be thinking ok you will know that these things in advance, but this is not always the case, this bug appeared, got mainstream media attention.

Overflow - Problem from real life

- In December 2014, The song "Gangnam Style" by the artist PSY was the most played video on Youtube (at that time) and reached 2.1 billion views.
- Youtube stored the view numbers of a video as an **int**, therefore, once it reached the max value of int which is 2,147,483,647 it **overflowed** and switched into negative number of views.
- So, they had to change the value stored the views variables into **long**.



https://www.vox.com/2014/12/3/7326945/gangnam-style-got-so-many-views-that-it-nearly-broke-youtube

- Let's apply the same solution to our code.

Question 2, Expansion 2 – Overflow

```
public static long factorial(long n) {
        if (n < 0){
                return -1;
        }
        long ans = 1;
        for (long i = 1; i <= n; i++) {
                ans *= i;
        }
        return ans;
}
```

Question 2, Expansion 2 – Factorial

- Let's test our function after the change

```
public static void main(String [] args) {
        System.out.println(factorial(1));
        System.out.println(factorial(0));
        System.out.println(factorial(5));
        System.out.println(factorial(-1));
        System.out.println(factorial(20));
}
```

Expected output:

1
1
120
error of sort
2432902008176640000

Actual output:

1
1
120
-1
2432902008176640000 // changed

Question 3, Expansion 3 – Binomial coefficient

- The binomial coefficient, often referred to as "n choose k" or "combinations", is a fundamental concept in combinatorics. It represents the number of ways to choose
- k items from a set of n items without replacement and without order.
- Where:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

- Design a program which implements the formula mentioned above which receives the values for n and k as command line arguments and passes them to functions

```
% java Binomial 5 2
    10
% java Binomial 15 6
    5005
```

Question 3, Expansion 3 – Solution

```
public class BinomialCoefficient {
    public static void main(String[] args) {
        long n = Integer.parseInt(args[0]); //GOOD NAME DUE TO FORMULA
        long k = Integer.parseInt(args[1]); //GOOD NAME DUE TO FORMULA
        System.out.println(binomial(n, k));
    }
    public static long binomial(long n, long k) {
       if (n < 0 || k < 0){
           return -1;
       }
       long numerator = factorial(n);
       long denoPart1 = factorial(k);
       long denoPart2 = factorial(n - k);
       return numerator / (denoPart1 * denoPart2);
    }

    public static long factorial(long n) {
      if (n < 0){
          return -1;
      }
      long ans = 1;
      for (long i = 1; i <= n; i++) {
          ans *= i;
      }
      return ans;
    }
}
```

Question 3, Expansion 4 – Students Union

- Consider the following problem.
- You have a class of *n* students. *k* of these students need to participate in the student union. *l* of the students in the union need to be part of the students' council.
- In how many ways can you choose a student union and council?
- The answer is purely combinatorial:

$$\binom{n}{k}\binom{k}{l}$$

- Write a program which:
  - Reads n,k,l as command line arguments. Such that n > k and k > l.
  - Prints the number of ways a students union/council can be assembled.
- Examples:

```
% java StudentsUnion 10 5 2
    2520
% java StudentsUnion 15 8 4
    450450
```

Question 3, Expansion 4 - Solution

```
public class StudentsUnion {

    public static void main(String[] args) {
        long n = Integer.parseInt(args[0]); //GOOD NAME DUE TO FORMULA
        long k = Integer.parseInt(args[1]); //GOOD NAME DUE TO FORMULA
        long l = Integer.parseInt(args[2]); //GOOD NAME DUE TO FORMULA
        System.out.println(binomial(n, k) * binomial(k, l));
    }

    public static long binomial(long n, long k) {
        if (n < 0 || k < 0){
             return -1;
        }
        long numerator = factorial(n);
        long denoPart1 = factorial(k);
        long denoPart2 = factorial(n - k);
        return numerator / (denoPart1 * denoPart2);
    }

    public static long factorial(long n) {
        if (n < 0){
             return -1;
        }
        long ans = 1;
        for (long i = 1; i <= n; i++) {
            ans *= i;
        }
        return ans;
    }
}
```

Overloading

- Consider the following, System.out.println(), is a function.
- How come you can print a String and you print an int with the same System.out.println() function? Since the signature of the function supposed to be different?
- **Definition**: Functions will be overloading functions if they share the same name under the same scope.
- Two functions can have the same name if and only if they have some difference in signature, which means a difference their parameters. (type, number, order).
- **Note: Conventionally**, overloading functions usually do the same operation with slight adjustment of type or functionality, but generally it is the same.

Question 4 – Substring

- You learned the function str.substring in the lectures, the function returns a part of a string based on the indices given.
- Build a 3 variations the function substring.
  - Receives a String and 1 integer number (given beginning to end of string)
  - Receives a String and 2 integer numbers (given beginning to given end)
  - Receives a String and 1 integer number (begin of string to given end)
    - No input validation is required.
    - Use the same function name "subs" for each.
- Is there a problem?
- YES .
  - The 1st and 3rd functions have problem since they both receive 2 parameters, String and int.
  - We can flip the order of the parameters, but this will be uncomfortable for the user. So, what do we do?
  - Use a boolean.

Question 4 – Overloading Substring - Solution

A

```
public static String subs(String str, int start) {
        String res = "";
        for (int i = start; i < str.length(); i++) {
                res += str.charAt(i);
        }
        return res;
}
```

B

```
public static String subs(String str, int start, int end) {
        String res = "";
        for (int i = start; i < end; i++) {
                res += str.charAt(i);
        }
        return res;
}
```

C

```
public static String subs(String str, int index, boolean isStartWithIndex) {
        if (isStartWithIndex){
                return subs(str, index); // goes to A
        }
        return subs(str, 0, index); // goes to B
}
```

Question 4 – Usage

```
public class Substring {

    public static void main(String[] args) {
        String word = "Hello World";
        String word1 = subs(word, 3); // lo World
        String word2 = subs(word, 3, 7); // lo W
        String word3 = subs(word, 2, true); // llo World
        String word4 = subs(word, 2, false); // He
    }

    // Code from previous slide
}
```