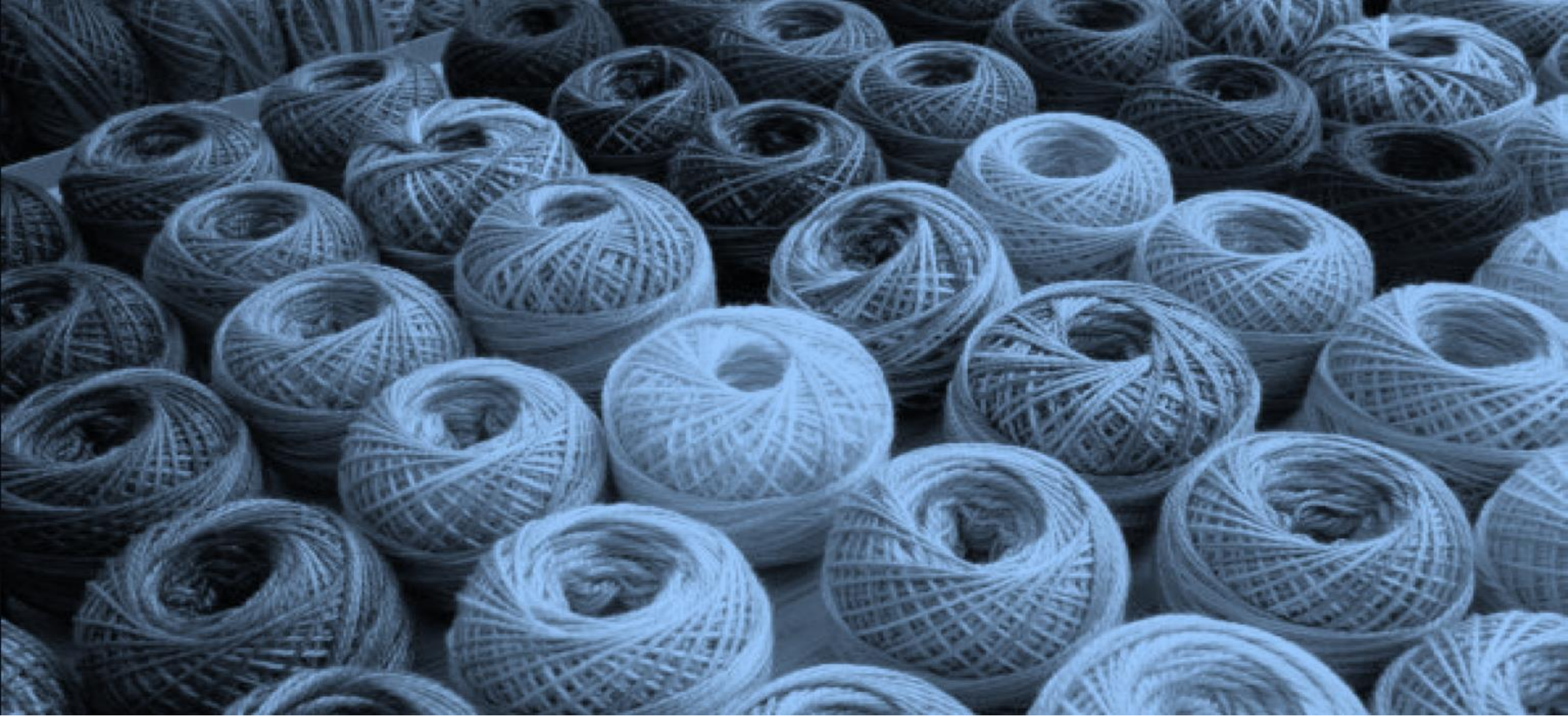


# Java Programming

CPT111 – Lecture 6
Erick Purwanto

Xi'an Jiaotong-Liverpool University
西交利物浦大学

CPT111 Java Programming

Lecture 6

# Static Methods 2 and String

# Welcome!

- Welcome to Lecture 6 – Static Methods 2 and Strings
- In this lecture we are going to learn about
  - More Static Methods
    - function definitions, implementations, calls
    - scope of variables
    - local variables vs global variables
  - Chars
    - char operations
  - Strings
    - primitive types vs reference types
    - string operations
    - string problem solving

# Part 1:  More Static Methods

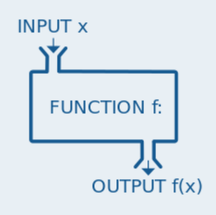
- In Week 5 we have learned about static methods or functions
    - we will explore more about it this week

# Recall Week 5:  Functions / Static Methods

- We want to avoid repeated codes
- We want to organize code as independent parts to share and reuse code to build bigger programs

```java
public class Lecture6Demo {
    public static void main(String[] args) {
        int x = 2, y = 5;
        if (x > y)
            System.out.println(x);
        else
            System.out.println(y);
        x = 10;  y = 8;
        if (x > y)
            System.out.println(x);
        else
            System.out.println(y);
    }
}
```

# Functions / Static Methods Definition



- Since all Java code is part of a class, we must define functions so that they belong to some class
- Functions are also called *static methods*

```java
public class Lecture6Demo {

    public static int larger(int x, int y) {
        if (x > y) {
            return x;
        }
        return y;
    }

    public static void main(String[] args) {
        System.out.println(larger(2, 5));
        System.out.println(larger(10, 8));
    }
}
```

visualize in https://pythontutor.com/java.html

# Functions / Static Methods

- Java function / static method:
  - takes zero or more input *parameters*
  - *returns* zero or one output value

- Examples seen: `Math.random()`, `Integer.parseInt()`

0 input, 1 return value

1 input, 1 return value

- Example 1:

2 input, 1 return value

```
public static int larger(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

# Functions / Static Methods

- Java function / static method:
    - takes zero or more input *parameters*
    - *returns* zero or one output value

- Example 2:

1 input, 0 return value

```
public static void printInts(int[] nums) {
    System.out.print("[");
    for (int i = 0; i < nums.length; i++) {
        if (i != nums.length-1)
            System.out.print(nums[i] + ", ");
        else
            System.out.print(nums[i]);
    }
    System.out.println("]");
}
```

# Function Implementation

- To implement a function
  - create a *name*
  - declare type and name of *parameter(s)*
  - specify *type* for *return* value
  - implement *body* of method
  - finish with *return statement(s)*

```
public static int larger(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

function name

parameter declarations

return type

body of `larger()`

return statement(s)

# Function Call

- To call a function
  - write the function name
    - if called from another class, write class name followed by . (e.g. `Math.random()` )
  - pass the *argument(s)* separated by commas in parentheses

```
System.out.println(larger(10, 8));
```

```
public static int larger(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

function name

arguments

# Scope

- The *scope of a variable* is the part of the program that can refer to that variable by name
    - the scope of the variables declared in a block of statements is limited to the statements in that block
    - in particular, the scope of a variable declared in a static method is limited to that method's body
        - therefore, we cannot refer to a variable in one static method that is declared in another
        - we call this *local variable*

# Scope Example

- The *scope of a variable* is the part of the program that can refer to that variable by name

```
public static void printInts(int[] nums) {
    System.out.print("[");
    for (int i = 0; i < nums.length; i++) {
        if (i != nums.length-1)
            System.out.print(nums[i] + ", ");
        else
            System.out.print(nums[i]);
    }
    System.out.println("]");
}

public static void main(String[] args) {
    int[] arr = {1, 2, 3};
    printInts(arr);
}
```

scope of nums

scope of i

scope of args

scope of arr

# In-Class Quiz 1  Method and Local Variables

- What is the output of the following Java program:

```java
public static void negate(int a) {
    a = -a;
}

public static void main(String[] args) {
    int a = 5;
    System.out.println(a);
    negate(a);
    System.out.println(a);
}
```

○ 5
5

○ 5
-5

○ -5
5

○ -5
-5

# Never Use Global Variables

- Best practice: declare variables so as to *limit their scope*

```
public class GlobalVariableDemo {

    public static int x;   // global variables
    public static int y;   // bad practice, don't do this

    public static int larger() {
        if (x > y) {
            return x;
        }
        return y;
    }

    public static void main(String[] args) {
        x = 2;
        y = 5;
        System.out.println(larger());
    }
}
```

# Method Signature

- *Method signature* of a method consists of
  - method name
  - parameter types

method signature

```java
public static int larger(int x, int y) {
    if (x > y) {
        return x;
    }
    return y;
}
```

# Overloading

- Static methods whose signatures differ are different static methods
- Using the *same name* for two static methods whose *signatures differ* is known as *overloading*

```
public static int larger(int x, int y) {
    if (x > y)
        return x;
    return y;
}
```

can you implement larger(int, int, int) using larger(int, int)?

```
public static int larger(int x, int y, int z) {
    int max = x;
    if (y > max) max = y;
    if (z > max) max = z;

    return max;
}
```

more in future lectures

# Part 2: Char and String

- We continue our lecture today on `char` and `String`
  - and to solve problems using their supported operations

# Primitive Data Type vs Reference Data Type

- A *data type* is a set of values and a set of operations defined on those values
- We have been learning the ***primitive** data types* that are built in Java
    - 8 primitive types: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, **`char`**
- Everything else, including arrays and **`String`**, is *not* a primitive type but rather *a **reference** type*
    - we will see later that when we declare a reference type in a variable, we store not the data, but the reference / address / pointer of the data
- In Part 2 of today's lecture, we will learn more about `char` and consider `String` as a reference type, and also learn about string processing

# Char

- A char is an alphanumeric character or symbol, enclosed in single quote ' '

```
char firstLetter = 'A';
char five = '5';
char newLine = '\n';
```

# Char characters

- A char is a Unicode character
  - it is effectively an integer,  ranges between 0 to 65,535 (inclusive)
  - it can be compared just like an integer,  with `==  !=  <  <=  >=  >`

```
char firstLetter = 'A';

firstLetter == 65;
```

# Char Testing

- We can use `Character` wrapper class static methods to test a character:
  - `boolean isLetter(char c)` is c a letter?
  - `boolean isDigit(char c)` is c a digit?
  - `boolean isWhitespace(char c)` is c white space?
  - `boolean isUpperCase(char c)` is c an uppercase?
  - `boolean isLowerCase(char c)` is c a lowercase?

```
char firstLetter = 'A';
char five = '5';
char newLine = '\n';

Character.isLetter(firstLetter);
Character.isDigit(five);
Character.isWhitespace(newLine);
Character.isUpperCase(firstLetter);
Character.isLowerCase(firstLetter);
```

# Uppercase, Lowercase, toString

- We can use `Character` class' static methods to change a character:
  - `Char toUpperCase(char c)` to uppercase
  - `Char toLowerCase(char c)` to lowercase
  - `String toString(char c)` to a `String` object consist of one character

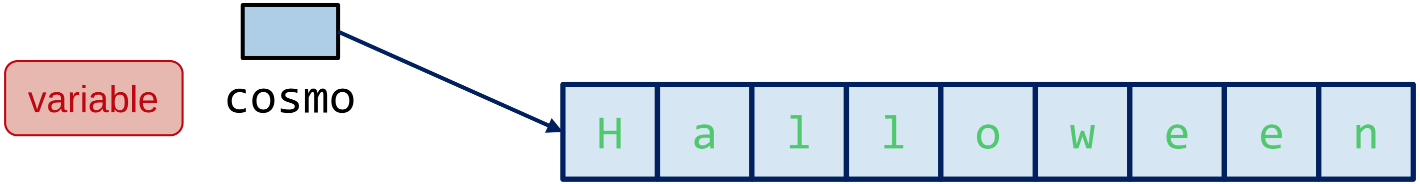
```
char firstLetter = 'A';
char smallA = 'a';

Character.toUpperCase(smallA);
Character.toLowerCase(firstLetter);
String oneLetterStr = Character.toString(firstLetter);
```

# Java String

- A *Java string* is a sequence of characters (chars), like the word "Halloween", or a sentence "A corpse is talking."
- Create a Java string by writing its chars between double quotes " "

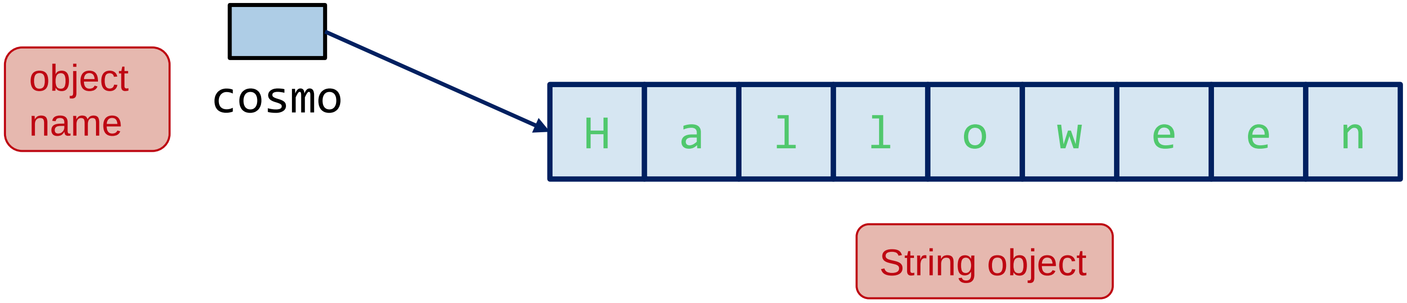
```
String cosmo = "Halloween";
```



# String Class and Object

- String is a *class* or a *data type* in Java, its name is capitalized
- Create a String *object* by constructing a new *object*
  - use the keyword new to call a *constructor*
  - use *class* or *data type name* to specify type of object

```
String cosmo;
cosmo = new String("Halloween");
```



object name

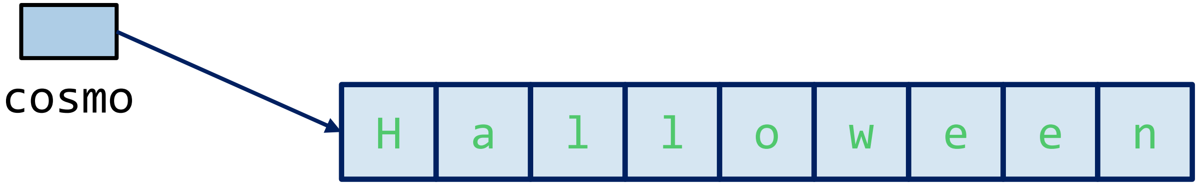
cosmo

String object

# Reference Variable Declaration

- When we declare a variable of any reference type such as String, Java allocates a box of 64 bits, no matter what type of object
  - the 64-bit box contains *not* the data about the string, but instead the *address* of the string in memory

```
String cosmo;
cosmo = new String("Halloween");
```



visualize in https://pythontutor.com/java.html

# Concatenation

- Plus + operator between strings concatenate them together to make a new, bigger string
  - chars of the first string put together with chars of the second string
  - works with strings stored in variables too

```
String cosmo = "hallo" + "ween";

String bang = "bang";
String exclamation = "!!";
String makima = bang + exclamation;
```

# Length and Method Call

- The length of a string is the number of chars in it
- A *method* called `length()` on a string returns its length
- To call a method / to apply an *operation*:
  - use object name to specify which object
  - use the *dot operator* to indicate that an operation is to be applied
  - use a method name to specify which operation

```java
String cosmo = "Halloween";
int len = cosmo.length();
System.out.println(len);
```

# In-Class Quiz 2  Empty String and Length

- What is the output of the following Java program:

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

    String s = "";
    int len = s.length();
    System.out.println(len);

}
```

○ nothing is printed

○ 0

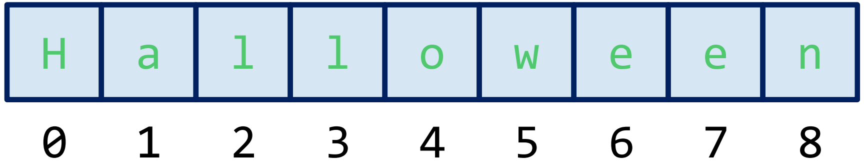
○ 1

○ 2

○ 3

# Index Number

- The chars in a string are identified by *index numbers* `0`, `1`, `2`, ...
    - leftmost char is at index `0`
    - last char is at index `length-1`



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| H | a | l | l | o | w | e | e | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

# Substring 1

- `substring()` method picks out a part of string using index numbers to identify the desired part
- `substring(int start)` returns a new string made of the chars *starting at* index `start` and continue until the end of the string

| H | a | l | l | o | w | e | e | n |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
String cosmo = "Halloween";
String ss1 = cosmo.substring(1);
String ss2 = cosmo.substring(2);
```

# Substring 2

- Another version `substring(int start, int end)` returns a new string of the chars starting at index `start` up to but not including the end index
  - notice the length of the resulting substring can be computed by subtracting `end - start`

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| H | a | l | l | o | w | e | e | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
String cosmo = "Halloween";
String ss1 = cosmo.substring(2, 5);
String ss2 = cosmo.substring(8, 9);
```

# String Index Error (1)

- It is common to make mistakes with the index numbers fed into `substring()`
    - the valid index numbers for substring are `0`, 1, 2, ..., `str.length()`, so we need to be careful *not* to pass in numbers outside that range
        - the last number, `str.length()`, is one beyond the end of the string, used for the "***up to but not including***" index number end
- For example, suppose we want to take the first 5 chars of a string `str`
    - we may end up with String index out of bounds error for some `str`

```
String prefix = str.substring(0, 5);
```

# String Index Error  (2)

- To avoid out of bounds errors, add an if-statement to check the length of the string
    - don't assume that a string is long enough,  check the `length()` before calling `substring()`

```
if (str.length() >= 5) {
    prefix = str.substring(0, 5);
}
else {
    // do something else when length is < 5
}
```

# String Equality

- Use the `equals()` method to check if 2 strings are the same
  - `equals()` method is *case-sensitive*

same value

- Recall the == operator used to compare primitive types `int`, `double`, `char`
  - it *does not work reliably* with object types such as `String`

```
String cosmo = "Halloween";
String makima = "Bang";
if (cosmo.equals("Halloween")) {
    // correct use .equals() to compare Strings
}
if (cosmo == "Halloween") {
    // do not use == with Strings
}
cosmo.equals(makima);
```

# String Equality Not Case Sensitive

- There is a variant of `String` equality check called `equalsIgnoreCase()` that compares two strings while ignoring uppercase/lowercase differences

```
String cosmo = "Halloween";

cosmo.equals("Halloween");
cosmo.equals("halloween");
cosmo.equalsIgnoreCase("halloween");
```

# String Testing

- We can test with:

| | |
|---|---|
| `boolean isEmpty()` | is string an empty string? |
| `boolean contains(String substring)` | does string contain substring? |
| `boolean startsWith(String prefix)` | does string start with prefix? |
| `boolean endsWith(String postfix)` | does string end with postfix? |

```
String cosmo = "Halloween";

cosmo.isEmpty();
cosmo.contains("lowe");
cosmo.startsWith("hal");
cosmo.endsWith("ween");
```

# Uppercase and Lowercase

- We can change string:
  - `String toUpperCase()` to uppercase
  - `String toLowerCase()` to lowercase

```
String cosmo = "Halloween";

cosmo.toUpperCase();
cosmo.toLowerCase();
```

# For-Loop and String

- Use for-loop to iterate over the characters of a string
  - for example,  loop to hit each index number once:

```
for (int i = 0; i < str.length(); i++) {
    // do something to str at index i

}
```

# chatAt

- `char chatAt(int i)` returns the *character* at index `i`
  - for example,  print each character of `str` once:

```java
for (int i = 0; i < str.length(); i++) {
    // do something to str at index i
    System.out.println(str.charAt(i));
}
```

# indexOf

- `str.indexOf(String target)` method searches left-to-right inside `str` for the string `target`
  - it returns the index number where the target string is *first found*, or `-1` if the target is *not found*
  - case-sensitive
  - you may use this instead of a for-loop to iterate over and look for a string

```java
String cosmo = "Halloween";

int a = cosmo.indexOf("allo");
int b = cosmo.indexOf("e");
int c = cosmo.indexOf("allO");
```

# lastIndexOf

- There is also `lastIndexOf(String target)` method that searches for the `target` from right to left

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| H | a | l | l | o | w | e | e | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
String cosmo = "Halloween";

int d = cosmo.lastIndexOf("e");
```

# indexOf, lastIndexOf with fromIndex

- There are versions of `indexOf` and `lastIndexOf` that take a `fromIndex` to specify *where the search should begin*
  - `int indexOf(String target, int fromIndex)` : left-to-right, starts the search at the given `fromIndex`
  - `int lastIndexof(String target, int fromIndex)` : right-to-left, beginning at the given `fromIndex`
  - `fromIndex` does not actually need to be valid
    - if it is negative,  the search happens from the start of the string
    - if it is greater than the string length,  then -1 will be returned

- Using these,  we can write a loop to find *all* the instances of the target in a string

# Find all instances

- You may use indexOf to find all the instances of a target string, or just use the standard for-loop
- For example, we use indexOf in a loop to find all string "Hallo" in str
  - notice the start variable is used in the loop to keep resetting the starting point of the search

```
public static void findHallo(String str) {
    int start = 0;
    while (true) {
        int found = str.indexOf("Hallo", start);
        if (found != -1) {
            // found one, do something here
        }
        if (found == -1) break; // found none, need to stop looping
        start = found + 2;      // move start up for next iteration
    }
}
```

# String Problem Solving Example 1

- Write a method to capitalize the text in parenthesis
  - given an input string with exactly one pair of parenthesis

```java
public static String capitalParen(String str) {

    int left = str.indexOf("(");
    int right = str.indexOf(")");

    String sub = str.substring(left + 1, right);

    sub = sub.toUpperCase();

    String result = str.substring(0, left + 1) + sub +
            str.substring(right);

    return result;
}
```

# Trim, Split, Replace

- Other popular string methods:
  - `String trim()` this string with leading and trailing whitespace removed
  - `String replace(String a, String b)` this string with `a` replaced by `b`
  - `String replaceAll(String a, String b)` this string with all `a`'s replaced by `b`'s
  - `String[] split(String delimiter)` array of strings between occurrences of `delimiter`

```
String cosmo = "  ha haa halloween  ";

cosmo = cosmo.trim();
cosmo = cosmo.replaceAll("h", "H");

String[] words = cosmo.split(" ");
for (int i = 0; i < words.length; i++) {
    System.out.println(words[i]);
}
```

# String Problem Solving Example 2

- Write a method to validate a membership card id, with the following conditions:
  - Must be 13 characters long
  - Must be 4 blocks
  - Must be separated by hyphens
  - First character must be between A and D
    - can be upper or lower case
  - Other 3 blocks are 3 characters long
  - 2nd and 3rd blocks are all numbers
  - Final block must be 2 numbers and a letter
    - the letter must be between A and T

A-123-456-23A

A-187-267-111

c-542-223-11G

A-187-267+111

A-187-11G

E-123-456-23A

b-555-88-123T

# In-Class Quiz 3  String Validity Check

- Which one(s) do you think needs to be checked first?
  - □ Must be 13 characters long
  - □ Must be 4 blocks
  - □ Must be separated by hyphens
  - □ First character must be between A and D
  - □ Other 3 blocks are 3 characters long
  - □ 2nd and 3rd blocks are all numbers
  - □ Final block must be 2 numbers and a letter
  - □ The letter must be between A and T

A-123-456-23A

A-187-267-111

c-542-223-11G

A-187-267+111

A-187-11G

E-123-456-23A

b-555-88-123T

# Skeleton Code and Testing Cases

- Start by writing an empty method (e.g. returning `true`), and write the test cases in main method

```java
public static boolean checkMembership(String input) {

    // check conditions, any one unsatisfied, return false

    return true;
}

public static void main(String[] args) {
    String test1 = "A-123-456-23A";
    boolean valid = checkMembership(test1);
    System.out.println("Case 1 Expect: true Real: " + valid);
    String test2 = "A-187-267-111";
    valid = checkMembership(test2);
    System.out.println("Case 2 Expect: false Real: " + valid);
}
```

# Checking Input Length

- Return false if any of the checking fails
    - Checking input length

```
public static boolean checkMembership(String input) {
    // check length
    if (input.length() != 13) {
        return false;
    }






    // passing all checks
    return true;
}
```

# Checking Blocks and Delimiter

- Return `false` if any of the checking fails
  - Checking number of blocks and delimiter

```
public static boolean checkMembership(String input) {
    // check length
    if (input.length() != 13) {
         return false;
    }

    // check four blocks, separated by hyphens
    String[] blocks = input.split("-");
    if (blocks.length != 4) {
        return false;
    }

    // passing all checks
    return true;
}
```

you will continue and add more checking to solve this problem in **Lab Exercise 6.2**

# Thank you for your attention !

- In this lecture, you have learned:
  - to create functions / static methods
  - about `char` primitive data type
  - about `String` reference data type
  - to use Character and String methods to do problem solving

- Please continue to Lab 6 to complete Lab Tasks and Lab Exercises, and then solve
  - Exercise #6.1, #6.2 and
  - CW1 #6.1, #6.2