Day-5 JA-111 Batch

Topics

- **Java Beans, POJO, getters and setters**
- **Scanner class**
- **String, StringBuffer and StringBuilder class**

An Example of Pure Encapsulation

Consider the following code

```
class Human{
  int age;
}

class Demo{
  public static void main(String args[]){
    Human h = new Human();
    h.age = -215;
  }
}
```

Once object of human class is available then user of this object may assign any value. We know that -215 is an invalid value for human age but for java it is an int type variable so it have any value within the range of int type. This type of coding practice is not a recommended one because variables of class can be assigned with bogus values.

Solution: Use Encapsulation; follow these basics rules-

1. The class must be public
2. All the fields should be private
3. For each field there should be corresponding public getter and setter method.
4. It should have a zero argument constructor.
5. It may have parameterized constructor (it is not the minimum requirement)

Such kind of classes can be referred by multiple different terms like POJO (Plain Old Java Object), Java beans and DTO (Data Transfer Object). The precise discussion about each of the term is beyond the scope (as of now).

Such kind of classes is reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Advantages of JavaBean

- The JavaBean properties and methods can be exposed to another application.
- It provides an easiness to reuse the software components.

The better way to write Human class

```
public class Human {
  private int age;

  public Human() {
    age = 18;
  }
  public Human(int age) {
    if(age < 0 || age > 120){
      //put some error message and return
    }
    this.age = age;
  }

  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    if(age < 0 || age > 120){
      //put some error message and return
    }
    this.age = age;
  }
}
```

Taking input in java

1. Add following line outside the class, after package statement
   `import java.util.Scanner;`

2. Create an object of Scanner class; pass system.in as parameter while creating object
   `Scanner sc = new Scanner(System.in);`

3. Call the method(s) of Scanner class to take input
   - **byte nextByte():** Scans the next token of input as byte
   - **short nextShort():** Scans the next token of input as an short
   - **int nextInt():** Scans the next token of input as an integer
   - **long nextLong():** Scans the next token of input as an long
   - **float nextFloat():** Scans the next token of input as float
   - **double nextDouble():** Scans the next token of input as double
   - **boolean nextBoolean():** Scans the next token of input as boolean
   - **String next():** Scans the next token of input as single word String
   - **String nextLine():** Scans the next token of input as single line String

   No separate method to take character input so use following code to take input
   `sc.next().chatAt(0);`

   Tip: If user provides an incompatible/illegal value then all methods discussed in example given above throw InputMismatchException that is an unchecked exception (Run time error). InputMismatchException is defined in java.util package.

4. close the object of Scanner class using close() method
   `sc.close();`

An Example

```
package com.masai.sprint2;
import java.util.Scanner;

public class InputDemo {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter your first name: ");
    String name = sc.next();
    System.out.print("Enter age: ");
    int age = sc.nextInt();
    System.out.print("You are indian citizen (true/false): ");
    boolean isIndian = sc.nextBoolean();
    System.out.println("Your details are as follow-");
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Indian? " + isIndian);
    sc.close();
  }
}
```

Output:

```
Enter your first name: Karan
Enter age: 18
You are indian citizen (true/false): false
Your details are as follow-
Name: Karan
Age: 18
Indian? false
```

The String class

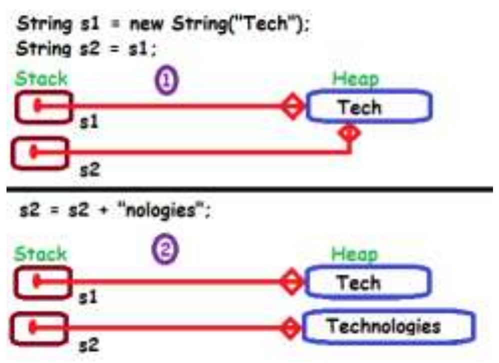
- Java does not implement string as a array of character instead of that java provides us a inbuilt class that is String.
- In java every object of String class is immutable that is it is impossible to change characters that a String object have but it is perfectly acceptable to update reference of String class. Each time we made changes in String literal we are creating a new object of String class. Previous String object remain unchanged. Although it is looking a serious restriction side but due to performance reason that restriction is imposed.
- Anything inside the double quotation ("") marks is considered as String literal, An object of String class can be created using new keyword also.
- String class implements CharSequence interface and declared as final hence it is not possible to create a subclass of String class.

What is the mean of Immutability?

Consider the following code

```
String s1 = new String("Tech");
String s2 = s1;
s2 = s2 + "nologies";
System.out.println(s1);      //Tech
System.out.println(s2);      //Technologies
```

Initially both s1 and s2 points to same object but as the `s2 = s2 + "nologies";` executes then new object will be created and s2 will start pointing to new object whose content is Technologies, s1 will keep continue to point to earlier object whose content is `Tech`.



Constructors of String class

- String(): Creates an empty String
- String(byte brr[]): Creates String from given byte array
- String(char chr[]): Creates String from given character array
- String(String s): Creates String from given String object
- String(StringBuilder sb): Creates String from given StringBuilder object
- String(StringBuffer sb): Creates String from given StringBuffer object

Methods of String class

- **char charAt(int position)**: returns character at the position from the calling string
- **int length()**: returns total number of characters in the calling string
- **byte[] getBytes()**: returns byte array from characters of String
- **char[] toCharArray()**: returns characters array from characters of String

An Example
```
String s1 = new String("Java");
System.out.println(s1.charAt(2));      //v
System.out.println(s1.length());       //4

byte brr[] = s1.getBytes();
for(byte b: brr)
System.out.print(b + " ");        //74 97 118 97

System.out.println();

char chr[] = s1.toCharArray();
for(char ch: chr)
System.out.print(ch + " ");       //J a v a
```

- **static String valueOf(int b)**: returns String representation of parameter int value
- **static String valueOf(float f)**: returns String representation of parameter float value
- **static String valueOf(Object d)**: returns String representation of parameter object value; It calls toString() method

An Example
```
String statement = "Yogesh of age " + 16 + " years has scored " + 92.25 +
"% in class XII";
```

is internally converted as

```
String statement = "Yogesh of age " + String.valueOf(16) + " years has
scored " + String.valueOf(92.25) + "% in class XII";
```

- **boolean equals(Object str)**: return true if parameter and calling object has same content (matching is case sensitive)
- **boolean equalsIgnoreCase(String str)**: return true if parameter and calling object has same content (matching is case insensitive)
- **int compareTo(String str)**: return 0 if parameter and calling object has same content (matching is case sensitive); otherwise return the difference of first character mismatch; otherwise if no character mismatch but any String exhausted then this.length() - str.length()
- **int compareToIgnoreCase(String str)**: Same as compareTo but matching is case insensitive
- **boolean startsWith(String str)**: returns true if calling String starts with str
- **boolean endsWith(String str)**: returns true if calling String ends with str

An Example

```
System.out.println("Java".equals("Java"));  //true
System.out.println("Java".equals("java"));  //false
System.out.println("Java".equalsIgnoreCase("java"));   //true

System.out.println("Java".compareTo("Java"));     //0
System.out.println("Java".compareTo("java"));     //-32
System.out.println("Java".compareToIgnoreCase("java"));     //0
System.out.println("java".compareTo("Java"));     //32
System.out.println("javaw".compareTo("java"));    //1
System.out.println("java".compareTo("javaw"));    //-1

System.out.println("Java".startsWith("Ja"));      //true
System.out.println("Java".startsWith("va"));      //false
System.out.println("Java".endsWith("va"));  //true
System.out.println("Java".endsWith("Ja"));  //false
```

- **int indexOf(char ch)**: returns the first matching index of string specified by parameter, return -1 if character not found.
- **int indexOf(String str)**: returns the first matching index of string specified by str, return -1 if str not found.
- **int lastIndexOf(int ch)**: returns the last matching index of character specified by ch, return -1 if ch not found.
- **int lastIndexOf(String str)**: returns the last matching index of string specified by str, return -1 if str not found.

An Example

```
System.out.println("bluetooth".indexOf('o'));     //5
System.out.println("bluetooth".lastIndexOf('o'));//6

String temp = "value has a value only if the value is valued";
System.out.println(temp.indexOf("value"));  //0
System.out.println(temp.lastIndexOf("value"));    //39
```

- **String substring(int startIndex)**: returns substring starts from startIndex to the end of String
- **String substring(int startIndex, int endIndex)**: returns substring starts from startIndex to the endIndex - 1 of String
- **String replace(char original, char replacement)**: returns a new String in which the original character is replaced with replacement character for the calling string
- **String trim()**: returns a new String with truncating leading and trainling spaces from the calling string

- **String toLowerCase()**: returns a new String with all characters in lowercase of calling string
- **String toUpperCase()**: returns a new String with all characters in lowercase of calling string

An Example
```
System.out.println("bluetooth".substring(4));     //tooth
System.out.println("bluetooth".substring(0, 4)); //blue
System.out.println("Hook".replace('H', 'L'));     //Look
System.out.println("  java   ".length());   //9
System.out.println("  java   ".trim().length()); //4
System.out.println("Java".toUpperCase());   //JAVA
System.out.println("Java".toLowerCase());   //java
```

How String literals and String objects are different?
When a String is created by writing the content in the double quoted mark but without new keyword then it is string literal. The string literal is created in the constant pool (It is memory area inside the heap). All literals in the constant pool are distinct yet multiple references can point to same literals.

When a String object is created using new keyword then it is string object. The string object is created in the heap. Each time you are using new keyword, you are forcing JVM to create new object no matter content with same object already exists or not.

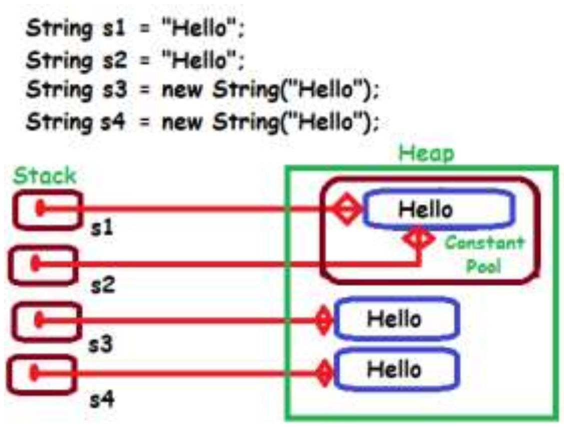
An Example
```
String s1 = "Hello";
String s2 = "Hello";

String s3 = new String("Hello");
String s4 = new String("Hello");

System.out.println(s1 == s2);    //true
System.out.println(s1.equals(s2));     //trues

System.out.println(s2 == s3);    //false
System.out.println(s2.equals(s3));     //true

System.out.println(s3 == s4);    //false
System.out.println(s3.equals(s4));     //true
```



```
String s1 = "Hello";
String s2 = "Hello";
String s3 = new String("Hello");
String s4 = new String("Hello");
```

StringBuffer & StringBuilder class
- Both classes provides for mutable sequence of characters.
- Both classes are final so they cannot be inherited.

- Both implements CharSequence interface and declared as final hence it is not possible to create a subclass of os both classes.
- StringBuilder class is not thread-safe but StringBuffer is thread-safe.
- StringBuilder class is faster than the StringBuffer class

Some method to mutate the StringBuffer and the StringBuilder object are as follow-

- **int capacity()**: Returns capacity(Number of character) that a StringBuffer object can hold
- **void setCharAt(int position, char ch)**: Set character to the position specified by argument ch and position respectively
- **StringBuffer reverse()**: Simply reverse content of calling StringBuffer object
- **StringBuffer insert(int index, String str)**: Insert str in calling StringBuffer object at the position specified by index
- **StringBuffer append(String str):** Append String object str at the end of calling StringBuffer object
- **StringBuffer delete(int start, int end)**: Delete character sequence from calling StringBuffer object, start deleting from startIndex to endIndex
- **void setLength(int len)**: Set length of StringBuffer object specified by argument len, if length is greater than number of characters than null characters are inserted. If length is less than number of characters than characters beyond the length will be lost.
- **StringBuffer deleteCharAt(int location)**: Delete a character from calling StringBuffer object specified by argument location

An Example

```
StringBuffer sb = new StringBuffer("Technology");
System.out.println(sb.length()); //10
System.out.println(sb.capacity());    //26
sb.setCharAt(0, 't');
sb.deleteCharAt(9);
sb.append("ies");
System.out.println(sb);     //technologies
sb.insert(9, 'y');
sb.delete(10, sb.length());
System.out.println(sb);     //technology
```

You Activity

Take you name input, convert the same to the array and the find total vowels and consonants on you name.