# Object Oriented Programming CS F213

J. Jennifer Ranjani
email: jennifer.ranjani@pilani.bits-pilani.ac.in
Chamber: 6121 B, NAB
Consultation: Appointment by e-mail

**BITS** Pilani
Pilani Campus

# -Packages
# -Downcasting
# -Strings

BITS Pilani
Pilani Campus

## Solution to Take Home Exercise given in the Previous Class

```
class Printable{
private interface Showable{
void show();}

class ShowClass implements
    Showable  {
public void show() {
System.out.println("Within Show");}
}
public void print() {
ShowClass s = new ShowClass();
s.show();
System.out.println("Within Print"); }
}
```

```
public class test {
public static void main(String[]
    args) {
Printable t = new Printable();
t.print();
}
}
```

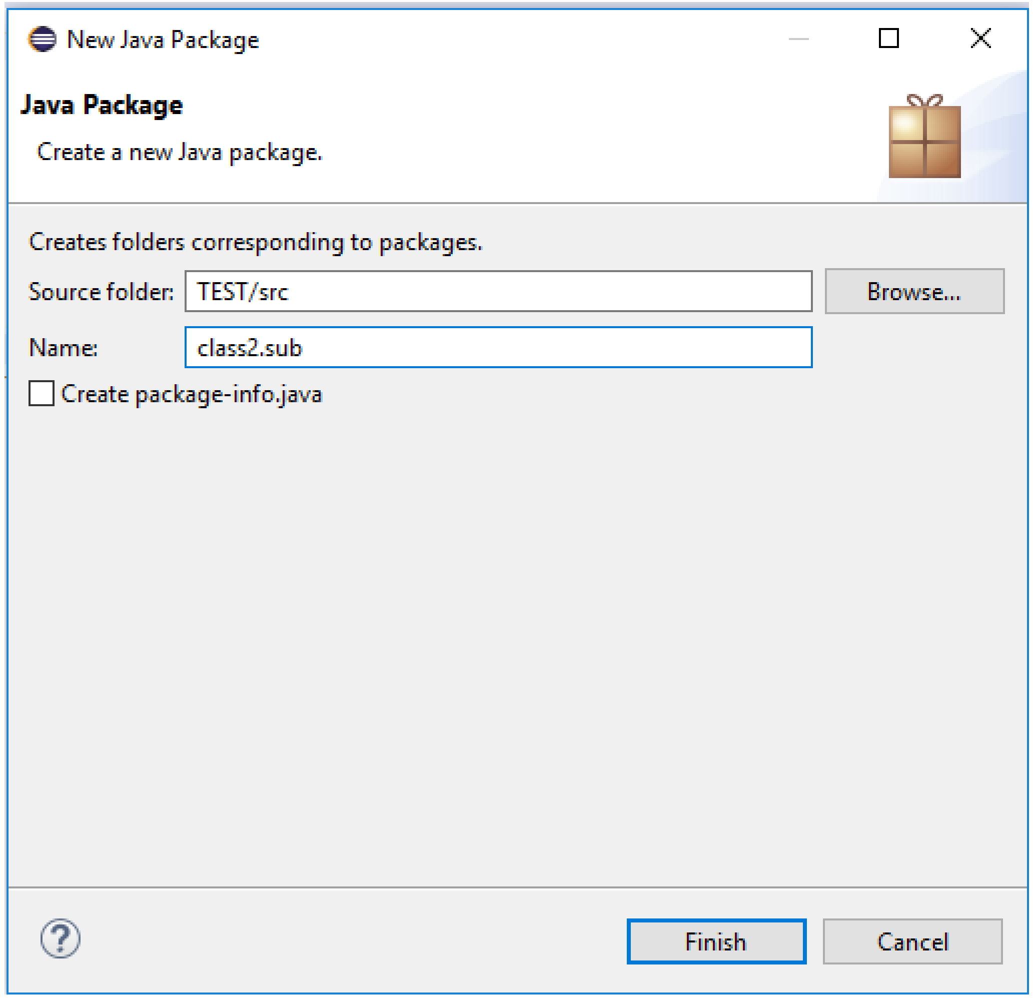
innovate achieve lead

BITS Pilani
Pilani Campus

## Packages

# Create a package & sub package

Project → New → Package


New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

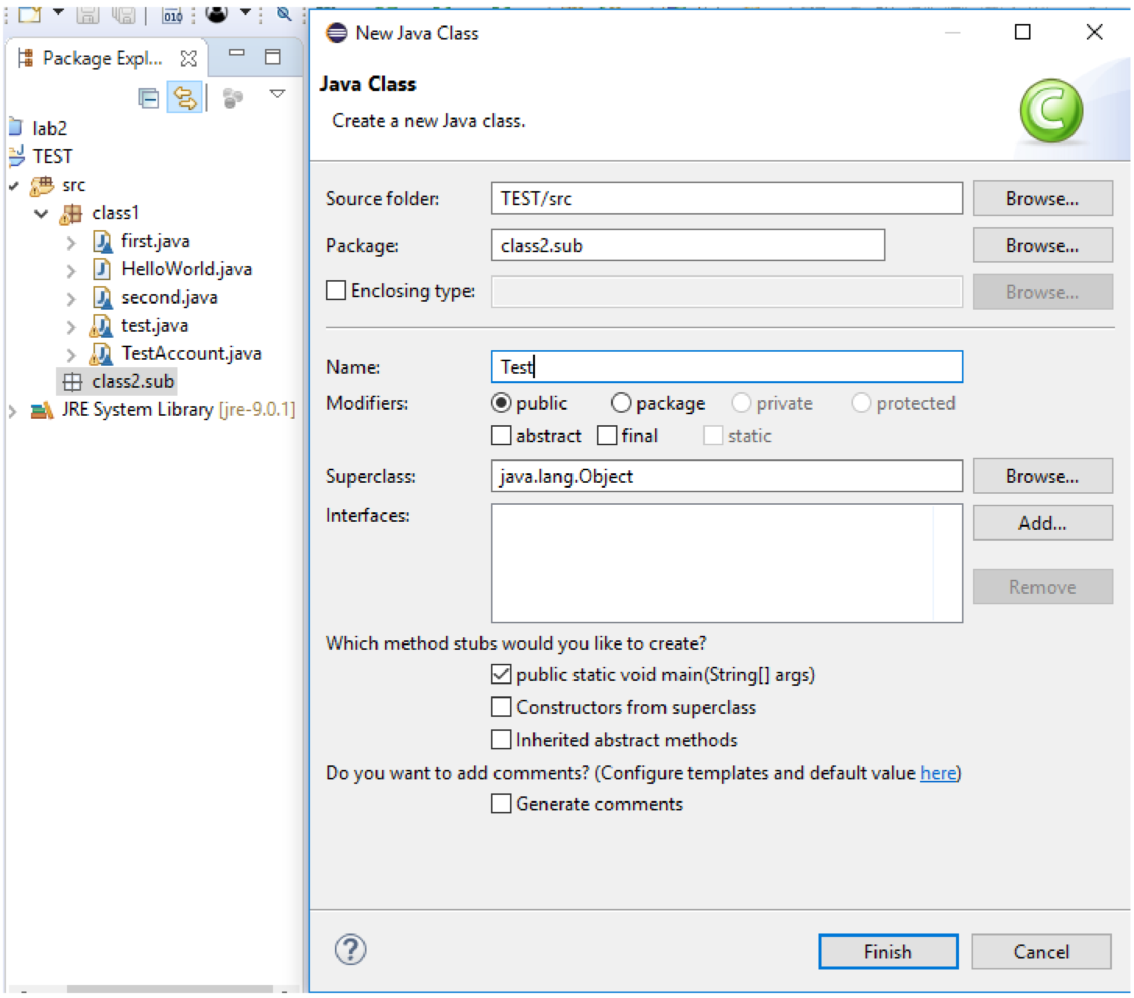
Source folder: TEST/src

Name: class2.sub

☐ Create package-info.java

# Create a class within the package

Package → New → class



New Java Class

**Java Class**

Create a new Java class.

Source folder: TEST/src

Package: class2.sub

[ ] Enclosing type:

Name: Test

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[ ] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

## Class within the package

```
package class2.sub;

public class Test {

public static void main(String[] args) {
// TODO Auto-generated method stub

}

}
```

# Importing a package

```
package class1;

public class HelloWorld
{
 public void show() {
  System.out.println("Within class
     1's show");
  }
}
```

```
package class2.sub;
import class1.*;

public class Test {

public static void main(String[]
     args) {
HelloWorld h = new HelloWorld();
h.show();

}

}
```

# Importing a class

```
package class1;

public class HelloWorld
{
 public void show() {
  System.out.println("Within class
    1's show");
  }
}
```

```
package class2.sub;
import class1.HelloWorld;

public class Test {
public static void main(String[]
     args) {
HelloWorld h = new HelloWorld();
h.show();
}
}
```

**Take Home Exercise: Learn how to execute the same code from the command prompt.**

# Access Modifiers

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|---|---|---|---|---|
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

innovate achieve lead

BITS Pilani
Pilani Campus

## Down casting

## Downcasting

- When subclass type refers to the object of the parent class
  - SavingsAccount sa = new BankAccount();  // compilation error

- If downcasting is done, no compilation error but ClassCastException is thrown at run time.
  - SavingsAccount sa =(SavingsAccount) new BankAccount();

# ‘instanceof’ operator

- It compares the instance with the type and returns true or false.
  - Account a1 = new Account();
  - System.out.println(a1 instanceof Account);
- If instanceof operator is applied on a variable that has null value, it returns false.
  - Account a1=null;
  - System.out.println(a1 instanceof Account);
- An object of subclass type is also a type of the parent class.

## Downcasting - Example

```
abstract class Animal {
    public void eat() {
        System.out.println("Eating...");
    }

    public void move() {

     System.out.println("Moving...");
    }

    public void sleep() {

     System.out.println("Sleeping...")
     ;
    }

}
```

```
class Dog extends Animal {
    public void bark() {
        System.out.println("Bow
     Bow!");
    }
    public void eat() {
        System.out.println("Dog is
     eating...");
    }
}

class Cat extends Animal {
    public void meow() {
        System.out.println("Meow
     Meow!");
    }
}
```

# Downcasting - Example

```
class AnimalTrainer {
   public void teach(Animal anim) {
      anim.move();
      anim.eat();

      if (anim instanceof Cat) {
         Cat cat = (Cat) anim;
         cat.meow();
      } else if (anim instanceof Dog)
    {
         Dog dog = (Dog) anim;
         dog.bark();
      }
   }
}
```

```
class test{
public static void main(String[]
    args) {
Dog dog = new Dog();
Cat cat = new Cat();

AnimalTrainer trainer = new
    AnimalTrainer();
trainer.teach(cat);
}
}
```

innovate achieve lead

BITS Pilani
Pilani Campus

# Strings

# Strings

- Java string is a sequence of characters. They are objects of type String.
- Once a String object is created it cannot be changed. Stings are Immutable.
- To get changeable strings use the class called StringBuffer.
- String and StringBuffer classes are declared final, so there cannot be subclasses of these classes.
- The default constructor creates an empty string.

```
    String s = new String();
```

## String creation

`String str = "abc";` is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

- If data array in the above example is modified after the string object str is created, then str remains unchanged.
- Construct a string object by passing another string object.

    String str2 = new String(str);

# String Constructors

- **String(byte[] byte_arr) – default character set (ASCII)**

```
byte[] b_arr = {74, 97, 118, 97};
String str =new String(b_arr);     // JAVA
```

- **String(byte[] byte_arr, Charset char_set)**

```
byte[] b_arr = {0x4a, 0x61, 0x76, 0x61};
Charset cs = Charset.forName("UTF-8");
String str = new String(b_arr, cs);
```

- ***Refer (List of character set supported by Java): https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html***

## String Constructors

- **String(byte[] byte_arr, String char_set_name)**

```
byte[] b_arr = {0x4a, 0x61, 0x76, 0x61};
String str = new String(b_arr, "UTF-8");
```

- **String(byte[] byte_arr, int start_index, int length)**
- **String(byte[] byte_arr, int start_index, int length, Charset char_set)**
- **String(byte[] byte_arr, int start_index, int length, String char_set_name)**

- **String(char[] char_arr)**
- **String(char[] char_array, int start_index, int count)**

BITS Pilani
Pilani Campus

## String Methods

## Length and Append

- The length() method returns the length of the string.

```
        System.out.println(“Hello World”.length());
// prints 11
```

- The + operator is used to concatenate two or more strings.

```
        String myname = “Harry”
        String str = “My name is ” + myname+ “.”;
```

- For string concatenation the Java compiler converts an operand to a String whenever the other operand of the + is a String object.

## String Methods-Character Extraction

- Characters in a string can be extracted in a number of ways.

- public char **charAt**(int index)
  - Returns the character at the specified index. An index ranges from 0 to length() - 1.

```
char ch;
ch = "Hello World".charAt(4);

String s1 = new String("Hello World");
ch=s1.charAt(4);
```

**Output:**
o

# String Methods-Character Extraction

- **public void getChars(int start, int end, char[] destination, int destination_start)**

```
s1 = "Hello World";
char ch[]=new char[20];
s1.getChars(0, 11, ch, 0);
```

- **public byte[] getBytes()**

- **public char[] toCharArray()**

```
s1 = "Hello World";
char ch[]=s1.toCharArray();
```

## String Methods

To compare two string objects

- **boolean equals( Object otherObj)**
- **boolean  equalsIgnoreCase (String anotherString)**
- **int compareTo( String anotherString):** Compares two string lexicographically.

```
s1 = "World";
s2 = "Hello";
p=s1.compareTo(s2);
```

**Output:**
15

- This returns difference s1-s2. If :

out < 0  // s1 comes before s2
out = 0  // s1 and s2 are equal.
out >0   // s1 comes after s2.

## String Methods

- **int compareToIgnoreCase( String anotherString)**
  - Compares two string lexicographically, ignoring case considerations.
- **String toLowerCase()**
  - Converts all characters to lower case
- **String toUpperCase()**
  - Converts all characters to upper case
- **String trim()**
  - Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.
- **String replace (char oldChar, char newChar)**
  - Returns new string by replacing all occurrences of *oldChar* with *newChar*

## String Methods

- **public boolean endsWith(String suf)**
  - Return *true* if the String has the specified suffix.

- **public boolean startsWith(String pre)**
  - Returns *true* if the String has the specified prefix

```
s1 = "Hello World";
s2 = "World";
System.out.println(s1.endsWith(s2));
```

# String Methods

- **public boolean regionMatches(int start_OString, String another, int start_AString, int no_of_char)**

- **public boolean regionMatches(boolean ignore_case, int start_OString, String another, int start_AString, int no_of_char)**

```
s1 = "HellO World";
s2 = "hello";
System.out.println(s1.regionMatches(1, s2, 1,
4));
```

## String Methods

- **String substring (int i) –** returns the substring from the ith index

```
s1 = new String("Hello World");
s2=s1.substring(4);
System.out.println(s2);
```

**Output:**
o World

- **String substring (int i, int j):** Returns the substring from i to j-1 index.

```
s1 = new String("Hello World");
s2=s1.substring(4,7);
System.out.println(s2);
```

**Output:**
o W

## String Methods

- **String concat( String str) –** Concatenates the string 'str' to the object invoking the method.

```
s1 = "Hello ";
s2 = "World";
s2=s1.concat(s2);
System.out.println("s1: :"+s1+"s2 :"+s2);
```

**Output:**
s1: :Hello s2 :Hello World

- **int indexOf (String s) –** returns index of the first occurrence of the specified string;
  - Returns -1 if not found

```
s1 = "World, Hello World, Hello";
s2 = "Hello";
p=s1.indexOf(s2);
```

**Output:**
7

## String Methods

- **int indexOf (String s, int i) –** returns index of the first occurrence of the specified string, starting at the specified index

```
s1 = "World, Hello World, Hello";
s2 = "Hello";
p=s1.indexOf(s2,8);
```

**Output:**
```
20
```

- **int lastIndexOf( int ch):** Returns the index within the string of the last occurrence of the specified string.

```
s1 = "World, Hello World, Hello";
s2 = "Hello";
p=s1.lastIndexOf(s2);
```

**Output:**
```
20
```

# String Methods

- **public int codePointAt(int index)**
  - returns the Unicode point of an index

```
s1 = "Hallo World";
p=s1.codePointAt(1);
```

**Output:**
97

- **public int codePointBefore(int index)**

- **public boolean contains(String str)**
  - Returns true if the invoking string object contains ‘str’

```
s1 = "Hello World";
s2 = “World";
System.out.println(s1.contains(s2));
```