

ANZ Java

Wednesday, February 1, 2023 10:59 PM

URL: [202302-anz-java https://1drv.ms/u/s!AknT1SrRpCz-wLEUhesLREpkzkkp4w?e=9QtUn0](https://1drv.ms/u/s!AknT1SrRpCz-wLEUhesLREpkzkkp4w?e=9QtUn0)

<http://tiny.cc/anz-java>

<http://t.co/xxyzbbz>

GIT: <https://github.com/vivekduttamishra/anz-java-202302>

C++

Thursday, February 2, 2023 8:46 AM

$C = C + 1$

- C with class
- new C

X++

C ++ --

Java

Thursday, February 2, 2023 8:54 AM

Java

- **Platform independent**
- **Architectural neutral**
- General purpose
- Object Oriented
- Multi-threaded
- Network
- **secured**
- robust
- high performance
- **interpreted**

programming language.

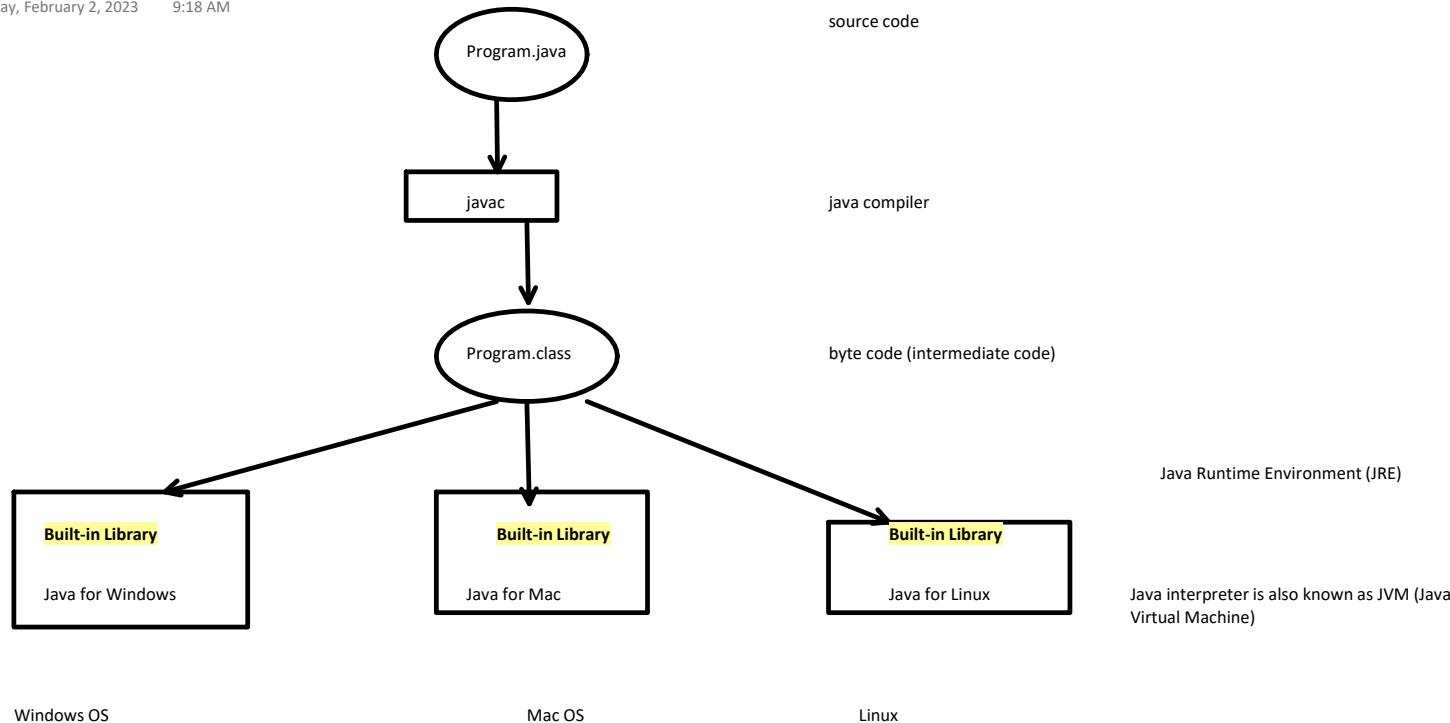
Java Promise

- Write Once, Run Anywhere
- No separate code for different OS/Hardware combination.



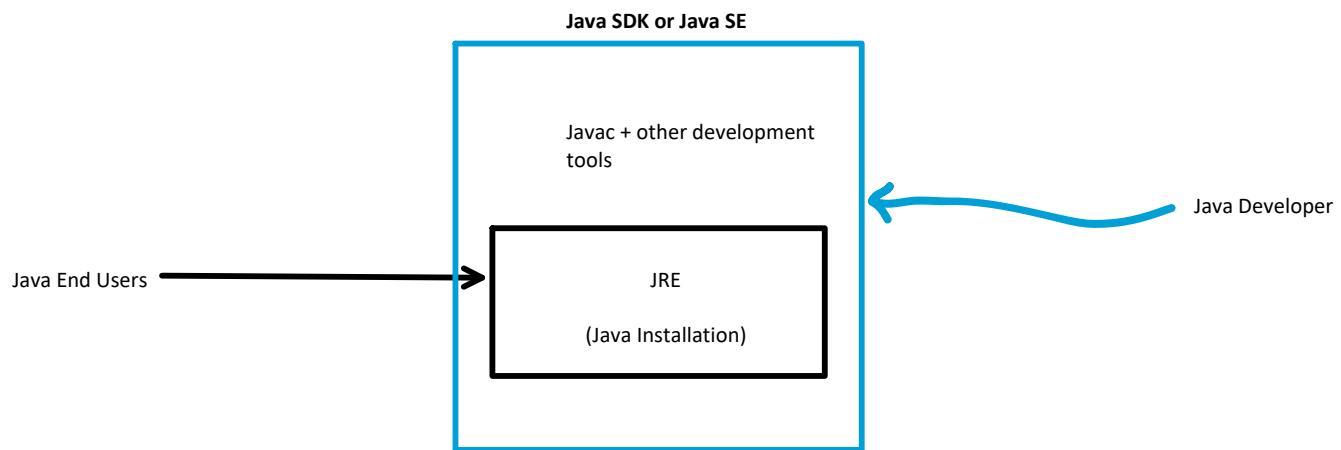
Java Program Flow

Thursday, February 2, 2023 9:18 AM



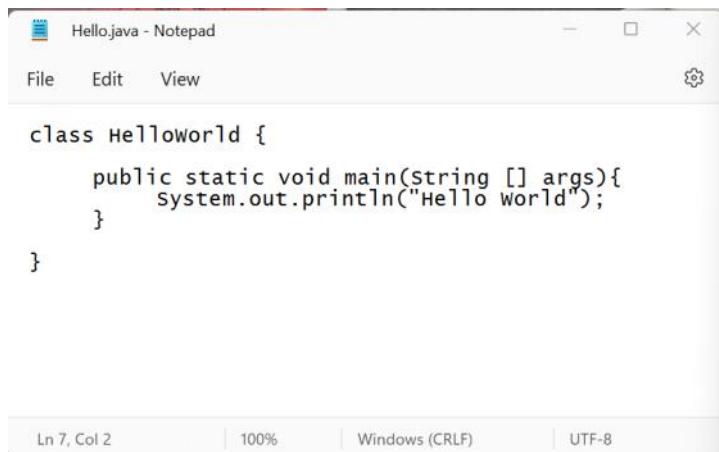
Installation and Bundles

Thursday, February 2, 2023 9:24 AM



Hello World

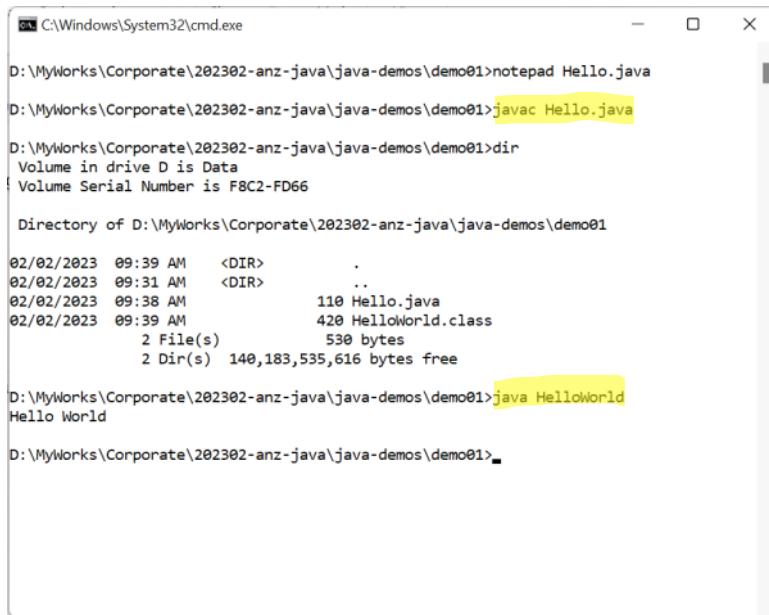
Thursday, February 2, 2023 9:59 AM



```
Hello.java - Notepad
File Edit View
class HelloWorld {
    public static void main(String [] args){
        System.out.println("Hello World");
    }
}

Ln 7, Col 2 | 100% | Windows (CRLF) | UTF-8
```

- Write a Hello.java
- We need
 - 1. A class
 - It can have any name we like
 - 2. main function
 - match exact signature
 - 3. print statement
 - a. match exact singature



```
C:\Windows\System32\cmd.exe
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01>notepad Hello.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01>javac Hello.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01>dir
Volume in drive D is Data
Volume Serial Number is F8C2-FD66
Directory of D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01
02/02/2023 09:39 AM <DIR> .
02/02/2023 09:31 AM <DIR> ..
02/02/2023 09:38 AM 110 Hello.java
02/02/2023 09:39 AM 420 HelloWorld.class
2 File(s) 530 bytes
2 Dir(s) 140,183,535,616 bytes free
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01>java HelloWorld
Hello World
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo01>
```

Step #1 compile

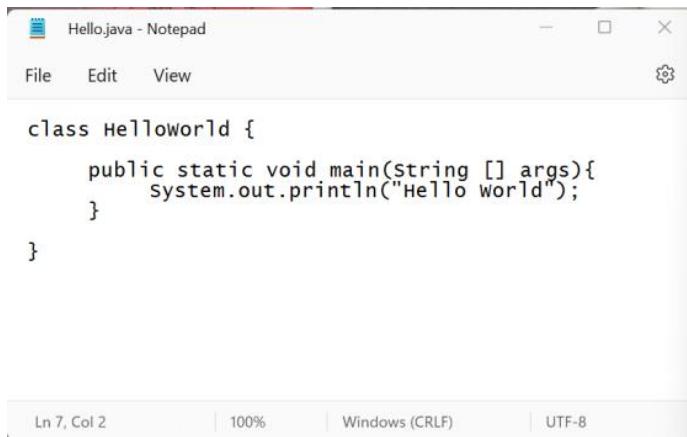
- we compile the source file.
- Here we use the full file name in the exact same case with extension
 - On success we get
 - A class file with same name as that of class
 - It may not be same as the file name

Step #2 run the program

- we run the class file that contains main
- name is case sensitve without suffixing .class

Basic Java

Thursday, February 2, 2023 10:34 AM



A screenshot of a Windows Notepad window titled "Hello.java - Notepad". The code inside is:

```
class HelloWorld {  
    public static void main(String [] args){  
        System.out.println("Hello world");  
    }  
}
```

The status bar at the bottom shows "Ln 7, Col 2" and "100%".

Naming Convention in Java

- Class Name
 - Pascal convention
 - Name should begin with upper case
 - If the name is a composite name it each word should begin with upper case
 - No underscores
 - Example
 - class Hello
 - class InterestCalculator
- Method Name/ Field Name / Variable Name
 - Camel case
 - Name should begin with lower case
 - In case of composite word each subsequent word should begin with upper case
 - avoid underscore
 - example
 - calculate()
 - calculateInterest()
 - period
 - interestRate
- package name
 - all lower case

Anatomy of Java Program

- A Java Program will have one or more classes
 - we need at least one class
- A class may have one or more methods (or functions)
 - Every program should have a "main" function
 - Every class doesn't need main.
- A Java Program is case sensitive.
 - You must be careful about the cases (upper case or lower case)
- Java Keywords
 - There are some special keywords that have special meaning in java
 - example
 - class
 - public
 - static
 - void
 - All keywords are in lower case
 - There are user defined words that represent
 - class name
 - method name
 - variable name
 - Example
 - HelloWorld
 - Few class names are pre defined by Java but are not keywords
 - String
 - System
 - out
 - println
 - main is special
 - It is created by user
 - Java expects you to create it
 - All user defined words can be in any case
 - You must use it in subsequent placed based on original definition.
 - We follow certain naming convention to avoid confusion

Simple Arithmetic Program

Thursday, February 2, 2023 10:50 AM

- Write a program to calculate sum of two numbers

```
ArithmeticApp01.java - Notepad
File Edit View
class Program{
    public static void main(String []args){
        int x=20;
        int y=30;
        int z=x+y;
        System.out.println(z);
    }
}
Ln 10, Col 25 100% Windows (CRLF) UTF-8
```

Variable

- To store a value of a particular type and refer it back we need to create a user defined name called variable
 - variable indicates that the value can change later.

```
int a= 20; //a is an integer that has current value 20.
char b= ''; // can hold international character set
double c=20.7; //can hold non-integer values

boolean d= true;
boolean e= 7>8; //false

• a variables value can change later
```

```
a = 30; //change the value to another value
a = a * 10; //change the value based on the previous value of same
variable
```

- You can't store wrong type of value in a variable

```
a="Hello World"; //can't store String in int variable
```

```
ArithmeticApp01.java - Notepad
File Edit View
class Program{
    public static void main(String []args){
        int x=20;
        int y=30;
        int z=x+y;
        System.out.println(z);
        x=false;
        System.out.println(x);
    }
}
```

Data Types

- to store the value in memory we need to create variables
 - variables are memory locations with specific name
 - they are associated with a particular type of value they can hold
- common types
 - int
 - integer
 - float
 - floating point (decimal numbers, single precision)
 - double
 - floating point decimal number, double precision
 - boolean
 - true/false
 - Other less used data types
 - char
 - a unicode char representation
 - represented as a single single quoted letter.
 - ◆ 'A'
 - ◆ '2'
 - ◊ '2' and 2 are different from each other
 - ◊ '2' doesn't possess arithmetic quality
 - byte
 - represents a single byte
 - short
 - short int
 - long
 - long int
 - String
 - String is a series of char to represent
 - word
 - sentence
 - It is double quoted
 - Note String begins with upper case S
 - It is a class and not a keyword
 - It is a predefined class created by Java team

Compatible and Incompatible type

- Few types are compatible if not same
- an int can be assigned to double without any information loss
 - Java allows this conversion automatically
 - implicit type conversion
- a double may be assigned to int with a loss of information (fraction part)
 - They are compatible but lossy
 - Java doesn't allow this conversion automatically

```
ArithmeticApp01.java - Notepad
int y=30;
int z=x+y;
System.out.println(z);
double d=x;
int u = d;
```

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac ArithmeticApp01.java
ArithmeticApp01.java:6: error: incompatible types: possible lossy conversion from double to int
 int u = d;
 ^
1 error
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>

- we can force such conversion by explicit type casting

```
int u = (int) d; //force convert value of 'd' in int before assign
```

- Note here 'd' remains double
- The value of d is converted to int and stored in u

Print A report including multiple variables

- what if we want to say
 - sum of 20 and 30 is 50
- Java allows "+" operator between string and anything
 - String + anything => string

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

        int x=20;
        int y=30;
        int z=x+y;
        System.out.println("sum of " + x + " and " + y + " is " + z);

    }
}
```

C:\Windows\System32\cmd.exe
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac ArithmeticApp01.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Program
sum of 20 and 30 is 50
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>

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

        int x=20;
        int y=30;
        int z=x+y;

        String output="sum of " + x + " and " + y + " is " + x + y;
        System.out.println(output);

        System.out.println( "sum of " + x + " and " + y + " is " + (x + y));
    }
}
```

C:\Windows\System32\cmd.exe
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac ArithmeticApp01.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Program
sum of 20 and 30 is 2030
sum of 20 and 30 is 50
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>

Statement vs Expression

- An expression can be a
 - simple value
 - arithmetic expression containing variable, constant and operators
- An statement
 - Always ends with a semicolon
 - A statement may be
 - declaring a variable
 - int x=20;
 - calling a method
 - System.out.println(x)
 - A method can take an expression as a parameter
 - It can't take statement as a parameter
 - We can't declare a variable as a method argument

`System.out.println(int x=20); //Not allowed.`

`System.out.println(x*20); //allowed`

White space

- Java considers blank space, tab and enter key or their combination as white space
- Wherever we can have a blank space or an operator, we can add any combination of white space
 - A statement may have multiple blank space, tab or even enter key
 - A statement or a expression may span to multiple lines
 - end is marked with semicolon

- valid statements may look like

```
int a=20; int b= a
+
30
/2 ;
```

- Note
 - statement 2 (declaration of variable b) begins in same line where first statement ends
 - second statement spans in 4 lines
 - It is acceptable

- **Exception to this rule**

- A string doesn't follow white space concept
- A string must end in the same physical line
- Invalid statement

```
String address = "A2 202, Ozone Evergreens,
Haralur Road,
Bangalore
560102 "
```

- To represent string with multiple line we use special combination characters to represent single character. This is known as escape sequences

- '\n' --> new line (also includes '\r')
- '\r' --> carriage return
- '\t' --> tab
- '\b' --> back space
- '\'' --> '
- '\"' --> "
- '\\' --> \

- To represent the above address properly

`String address = "A2 202, Ozone Evergreens,\nHaralur Road,\nBangalore\npin\t560102";`

- To represent a large string in source code we can use string concat

```
address= "A2 202,\n"+
"Ozone Evergreens,\n"+
"Haralur Road,\n"+
"Bangalore,\n"+
"pin\t560102";|
```

Java Operators

Thursday, February 2, 2023 12:01 PM

Operators	Meaning	Associative
()		inner to outer (right to left)
.		left to right
* , / , %, ~, !		left to right
+, -		left to right
<,>,<=,>=, ==, !=	Relational	left to right
&&	Boolean and	left to right
	Boolean or	left to right
=	assignment	right to left
+=		
-=		
*=		
?:		

- $20 * 30 / 40$
 - $20 * 30 = 600$
 - $600 / 40 = 15$

- $20 + 40 * 4$
 - $40 * 4 = 160$
 - $20 + 160 = 180$

- $(20 + 40) * 4$
 - $20 + 40 = 60$
 - $60 * 4 = 240$

Composite Assignment

- $x += y$
 - $x = x + y$
- $x *= y$
 - $x = x * y$

- $x = x + 1$
 - $x += 1$
 - $x ++$
 - $++x$

- $x = x - 1$
 - $x -= 1$
 - $x --$
 - $--x$

```
int x=20;  
x++; //21  
++x; //22
```

```
int y=5;  
int z = y++ * 10; //z will be 5* 10 =50, y will become 6 later
```

```
int k=5;  
int l = ++k * 10 ; // first k will become 6 then l will become 6*10 = 60
```

Integer Operations

```
int x=50;  
:-- ..--..
```

- an operation between 2 int always returns an int
 - It truncates (not rounds) to fractional part
- An operation involving at least one double makes the result double

```
int x=50;  
int y=4;  
  
double z= x/ y;  
  
System.out.println(z); //12.5
```

- an operation between 2 int always returns an int
 - It truncates (not rounds) to fractional part
- An operation involving at least one double makes the result double
 - $z = x / y;$
 - $x/y \Rightarrow 50/4 = 12.5 = 12$
 - $z = 12$
 - $z = 12.0$

- **How to get 12.5?**

- At least one operand should be double (or casted to double)

- Option#1

```
z= (double)x /y;
```

- Option#2

```
z= x*1.0/y;
```

Java Methods (functions)

Thursday, February 2, 2023 12:45 PM

- methods are independent reusable algorithm
- They have
 - name
 - return type (that can also be void if we don't return anything)
 - can take one or more parameters

A Method may represent a piece of executable code

```
C:\Windows\System32\cmd.exe

C:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
JavaMethod.java:12: error: non-static method greet() cannot be referenced from a static context
        greet(); //call the other method
               ^
1 error

C:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>
```

```
JavaMethod.java - Notepad

File Edit View

class Method01{
    void greet(){
        System.out.println("Hello Java World");
    }

    public static void main(String []args){
        greet(); //call the other method
    }
}
```

- A static main, can't call non-static greet
- main must be static
- We may also mark greet as static
- Why?
 - We will discuss later!

Working with multiple Methods

```

File Edit View

class Method01{

    static void greet(){
        System.out.println("Hello Java World");
    }

    public static void main(String []args){
        greet(); //call the other method
        greet(); // call again
        greet(); //and yet again
    }
}

C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Method01
Hello Java World
Hello Java World
Hello Java World
Hello Java World

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Method01
Hello Java World
Hello Java World
Hello Java World
Hello Java World

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Method01
Hello Java World
Hello Java World
Hello Java World
Hello Java World

```

Note:

- We have two methods in our program
 - greet
 - main
- Even though "greet" is the first method, program always begins with "main"
- "greet" will not work unless it is called explicitly
 - if main never calls it, it doesn't work. just exists
- main may call greet multiple times
- there can be more methods forming a chain
 - main calls method1
 - method1 calls method2
 - ...

What if we need to greet someone specific?

- we may pass the name of the person to be greeted as a parameter
- A parameter is like a variable that is created and assigned the value passed.

```

File Edit View

class Method01{

    static void greet(String name){
        System.out.println("Hello "+name+", welcome to Java World");
    }

    public static void main(String []args){
        greet(); //call the other method
        greet(); // call again
        greet(); //and yet again
    }
}

C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
JavaMethod.java:12: error: method greet in class Method01 cannot be applied to given types;
        greet(); //call the other method
               ^
required: String
found:   no arguments
reason: actual and formal argument lists differ in length
JavaMethod.java:13: error: method greet in class Method01 cannot be applied to given types;
        greet(); // call again
               ^
required: String
found:   no arguments
reason: actual and formal argument lists differ in length

```



```

class Method01{

    static void greet(String name){
        System.out.println("Hello "+name+", welcome to Java World");
    }

    public static void main(String []args){
        greet("Vivek"); //call the other method
        greet("Raheem"); // call again
        greet("Venu"); //and yet again
    }
}

C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Method01
Hello Vivek, welcome to Java World
Hello Raheem, welcome to Java World
Hello Venu, welcome to Java World

```

Note

- Here greet expects user to pass a String value
- That string value will be stored in a variable called name.
- greet method may use the name in their code
- But main is not passing the value for name and that is an error here
- Error
 - I couldn't find greet that doesn't take parameter

- Here we are calling greet multiple times with different values for name
- The supplied values are called arguments
- variable that is created to store argument is known as parameter
- Example
 - parameter is "name"
 - arguments supplied for "name" are
 - "vivek"
 - "rahim"
 - "venu"

Method chaining

```
File Edit View

class Method01{
    static void greet(String name){
        System.out.println("Hello " + name + ", welcome to Java world");
    }

    static void goodBye(){
        System.out.println("Good Bye everyone, see you soon");
    }

    public static void main(String []args){
        greetEveryone();
    }

    static void greetEveryone(){
        greet("Vivek"); //call the other method
        greet("Raheem"); // call again
        greet("Venu"); //and yet again
    }
}

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Method01
Hello Vivek, welcome to Java World
Hello Raheem, welcome to Java World
Hello Venu, welcome to Java World
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>
```

1. Program always begins with main()
2. main calls greetEveryone()
3. greetEveryone() calls greet() thrice with different parameters
4. no one calls goodBye() in the call chain that started with main
 - a. It never executes
 - b. It will not give any compile time error for being unused.
5. Physical order of method definition has no impact.

Method returning result

```
class Program{

    public static void main(String []args){
        int a = sumSquare(5,3);
        System.out.println(a);

        int b = sumSquare(4,6);
        System.out.println(b);
    }

    static int sumSquare(int a, int b){
        int c = a+b;
        return c*c;
    }
}

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod02.java
JavaMethod02.java:17: error: invalid method declaration; return type required
    static sumSquare(int a, int b){
           ^
1 error

D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac JavaMethod02.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>java Program
64
100
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>
```

Note

- sumSquare indicates that it is returning a value of type int.
- Before the function ends it must include a return statement with value that we need to return
- The returned may be used in the caller function as expression
 - assigned to a variable
 - included in a formula
 - int c=sumSquare(5,5)*10;
 - print directly
 - System.out.println(sumSquare(2,3));
- each method has its own set of variables
- main has
 - a
 - b
- sumSquare ahs
 - a
 - b
 - c
- It is possible that two different method has variables with the same name
 - They belong to different method and are unrelated
 - same name is just a co-incidence

Statements

Thursday, February 2, 2023 2:27 PM

- every statement ends with a semicolon
- a block of statement is wrapped in braces {}
- Java statements like if, while, for etc can take either a single statement or a block of statements

If statement

Important

```
if ( boolean_expression){  
    statement1;  
    statement2;  
}  
  
if(boolean_expression)  
    single_statement;
```

If - else

```
if( boolean_expression)  
    statement_or_block;  
else  
    statement_or_block
```

If -else if

```
if( condition1 )  
    do_this;  
  
else if(condition2)  
    do_this  
else if (condition 3)  
    do_this;  
else  
    do_this;
```

while loop

```
while( condition_is_true)  
    block_or_statement;
```

do-while

```
do{  
    one_or_more_statement;  
}while( condition_is_true);
```

Note

- do-while needs block marker even for a single statement

- do while executes a min of one time before it tests for condition

standard for loop

- similar to c/c++ etc

```
for( initialization; condition; reinitialization)  
    block or statement;
```

- for executes in the order
 1. runs initialization
 2. checks condition
 3. runs block or statement if condition is true else exists
 4. runs reinitialization
 5. repeats from step 2

```
for(int i=0; i<10; i++)
    greet();
```

- Note
 - Initialization can declare a new variable here.
 - All the three components are optional in for loop
 - You may or may not provide
 - initialization
 - ◆ if it is already done before for loop
 - condition
 - ◆ defaults to true
 - ◆ if not given it is like run for ever
 - re-initialization
 - ◆ if you are doing within the block
 - But the two semicolon inside for () is compulsory

- example for a run for ever for loop

```
for(;;)
    run_for_ever;
```

Examples

```
void countDown(int x){

    for(;x>0;x--){
        System.out.println(x);
    }
}
```

Example 2

```
void countDown(int x){

    for(;x>0;){
        System.out.println(x--);
    }
}
```

How to exit a loop without finishing

- sometimes we need to exit a loop (for/while/do-while) before its natural condition
 - we may have more than one condition and it may be complicated to put all in one place
- we can use "break" statement to exit the loop
- for example break the loop after you get three values that are divisible by 5 while counting in a range
- **IMPORTANT!**
 - **NEVER USE BREAK INSIDE A LOOP WITHOUT A CONDITION**

```
void countRange(int min, int max){

}
```

[Skipping a loop count](#)

- sometimes we may want to skip remaining statements of a loop under a given condition.
- We may use continue to denote that.
- continue should be conditional
- Let's skip every multiple of 2

Important!

- break and continue operates on the innermost loop in case of nested loop.
- you may need multiple breaks to come out of all the loops

[for-each style loop](#)

- will discuss later.

[switch statement](#)

[semantic](#)

```
switch(expression){  
    case value_1:  
        statement1;  
        statement2;  
        break;  
    case value_2:  
        statement1;  
        statement2;  
        return;  
  
    default:  
        statement;  
  
}
```

- A switch can take a expression that can be
 - number
 - string
- the value is matched to each case and and statement under the case is executed
- if no value matches the passed value it goes to default
- we should end each case with break or return
 - return exists the function
- we may use continue in switch case if it is present in some loop.
 - continue continues loop not switch

[90360 VIVEK](#)

Multiple classes

Friday, February 3, 2023 8:16 AM

- When we compile a source file it generates one .class file per class (not per file)

The screenshot shows two windows side-by-side. On the left is a Notepad window titled "FactorialApp.java - Notepad" containing the following Java code:

```
class App{
    public static void main(String[] args){
        int n=5;
        int fn=Math.factorial(n);
        //System.out.println("Factorial of "+n+" is "+fn);
        System.out.printf("Factorial of %d is %d\n", n , fn);
    }
}
class Math{
    static int factorial(int x){
        int fx=1;
        while(x>0)
            fx*=x--;
        return fx;
    }
}
```

On the right is a Command Prompt window titled "C:\Windows\System32\cmd.exe" showing the compilation and execution of the code. The command `javac FactorialApp.java` is run, followed by `java App`. The output shows the factorial of 5 is 120. A directory listing for the folder `D:\MyWorks\Corporate\202302-anz-java\java-demos\demo-set-02` is shown, including files `App.class`, `FactorialApp.java`, and `Math.class`.

How the Application is build with multiple source file?

- When we compile a source file say PermutationApp.java, java compiler finds it's dependency on class Permutation
- Now Java Compiler looks for a file Permutation.class
 - If present it uses the Permutation.class
- If Permutation.class is not present it looks for a source file with the same name
 - If present, it compiles the source file to get .class file
 - If it is not present, compilation aborts with error
 - IMPORTANT**
 - While it is not compulsory to have source file and class file with same name, it is good to have to assist the compilation process.
- If both source file and class file is available, compiler checks for the modification date to find which one is latest
 - In case source file is modified after last compilation, it is recompiled
- IMPORTANT:**
 - the source file is used only by java compiler and not by java runtime
 - Java runtime can't compile even if files are out of date.

Multiple Main class

```
C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo-set-02>java PermutationApp
5 P 3 = 60

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo-set-02>java Permutation
Help for Permutation:
Permutation.calculate(n,r)
Example: Permutation.calculate(8,3)=336

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo-set-02>
```

- We can have multiple main class in different classes
- The one which is invoked with Java command will be called
- Use Case
 - A Dictionary class can be used as
 - stand alone dictionary app
 - embedded in Word to spell check.

Object Oriented Program

Friday, February 3, 2023 8:56 AM

What is a Program?

Set of instructions given to computer to perform some task.

Furniture Shop

Friday, February 3, 2023 9:22 AM

Objects

- Furnitures
 - Chair
 - Material
 - Price
 - Table
 - Bed
 - List (Inventory)
 - List (Customer)
 - Invoice
- 
- Multiple Chairs with similar property and behaviors (purpose)
 - Each of them will have common set of elements like
 - Material
 - Price
 - They may also have different features like
 - recliner
 - drawer
 - (we will not talk about them,yet)

Creating Object, The Java Way (common in most languages)

- We need a class to represent the idea of an object
- A class will describe what an object will be like
 - It can be considered as template or blueprint
 - Why do we call it a class then? (pending question)
- To create an object we need a class (to describe it)

```
class Chair{  
}  
  
• Now we can create multiple chair objects  
  
Chair c1 = new Chair();  
Chair c2 = new Chair();
```

IMPORTANT!

- We, in most cases, would name our class as Singular
 - Chair, not Chairs
- A class is the description for a single Object
- Once we have the design description we can create multiple objects with same idea (class)

- Now a class can contain informations related to the object

```
class Chair{  
    int price=2000;  
}  
  
• Now our chairs can have price
```

```
Chair c1=new Chair();  
Chair c2=new Chari();  
  
System.out.println(c1.price); //2000;
```

- We can also change the price

```
c1.price=5000;  
System.out.println(c1.price); //5000;  
System.out.println(c2.price); //2000
```

Note

- class doesn't have the price variable
 - It has the definition of price which will belong to the chair object
- Both chair object will have their own price
 - so now we have to price variables
 - c1.price
 - c2.price
 - here 2000 is the default price that will be initially assigned to all chairs
 - each chair can individually change it

- An object can also have its own behavior or roles

```
class List{  
    int items:
```

- Here addItem is NOT a static method
 - Non static methods are referred as object level methods

```

class List{
    int items;
    void addItem( String item){
        items++;
        System.out.println("Item added");
    }

    int size(){
        return items;
    }
}

```

- Here addItem is NOT a static method
- Non static methods are referred as object level methods or instance methods
 - They belong to individual objects
- Most of your methods should be non-static
 - You are writing an object oriented program

- Now we can use these elements

```

List inventory = new List();

inventory.addItem("Chair");
inventory.addItem("Table");

List customers=new List();
customers.addItem("Vivek");

System.out.println( inventory.size()); //2
System.out.println(customers.size()); //1

```

Assignment 2.1

Friday, February 3, 2023 9:45 AM

- Create a List class
- Add the methods to
 - AddItem
 - RemoveItem
 - Size
- Test the methods with at least two list objects

Naming Convention

Friday, February 3, 2023 10:30 AM

```
class ClassList{  
    int items; //defaults to 0  
    void addItem(String item){  
        items++;  
    }  
    void removeItem(String item){  
        items--;  
    }  
    int countItems(){  
        return items;  
    }  
}
```

- This is a working code.
 - But a working code may not be equal to a good code
- Important considerations
 - Class Name doesn't need a Class Prefix
 - We Generally avoid prefix in any code
 - Between Prefix and Suffix prefer suffix
 - avoid both if possible
 - While addItem is a good name
 - Item is redundant suffix
 - in list add means addList
 - It can be avoided in this context
 - we don't need chairPrice and tablePrice in Chair and table class
 - Both can have price
 - meaning will be clear when we write
 - ◆ chair1.price
 - ◆ chair2.price
 - ◆ table1.price

Closer Look at the Objects

Friday, February 3, 2023 10:48 AM

```
public static void main(String []args){  
    List customerList=new List();  
    customerList.add("Vivek");  
  
    List furnitures=new List();  
    furnitures.add("Chair");  
    furnitures.add("Table");  
    furnitures.add("Bed");  
  
    System.out.printf("Total Customers: %d\n", customerList.count());  
    System.out.printf("Furnitures: %d\n", furnitures.count());  
  
    Chair c1=new Chair();  
    Chair c2=new Chair();  
  
    c1.price=3000;  
    System.out.printf("c1.price=%d\nc2.price=%d\n",c1.price,c2.price);  
  
    Bed b1=new Bed();  
    Inventory inventory=new Inventory();  
    Invoice invoice1=new Invoice();  
    Invoice invoice2=new Invoice();  
  
    Table t1=new Table();  
  
    System.out.println(t1);  
    System.out.println(b1);  
    System.out.println(inventory);  
    System.out.println(invoice1);  
    System.out.println(invoice2);  
}
```

```
Total Customers: 1  
Furnitures: 3  
c1.price=3000  c2.price=2000  
Table@33c7353a  
Bed@681a9515  
Inventory@3af49f1c  
Invoice@19469ea2  
Invoice@13221655
```

```
System.out.println(t1);  
System.out.println(b1);  
System.out.println(inventory);  
System.out.println(invoice1);  
System.out.println(invoice1.toString());  
System.out.println(invoice2);
```

List

- A list has three methods
 - add
 - remove
 - count
- It has a property
 - items
- They are interconnected for the same object
 - add increases items count
 - remove decreases the same field
 - count returns the result for the same
- The two lists are different from each other
 - add or customerList and add of furnitures are incrementing different "items" variable
- Similarly we have two Chair with their individual prices

Note

- We are here printing the entire Object and not some property of Objects
- Java internally prints an object as String with two components separated by "@"
 - Class Name of that object
 - A unique Id or hashCode generated for each individual object
 - By default they will be different for each object
 - Known as hashCode
- This information is also available by calling a special method present in all "objects" called toString
 - Internally System.out.println is implicitly calling toString of the current object
- the hashCode can be checked by using another special method hashCode()

- whenever we want to print an object, it internally calls the toString method

What if I want to print a different information for my object?

we can write our own toString method

The Default ToString Behavior

```
customerList List@548c4f57  
furntiures List@1218025c
```

After Adding our own `toString`

```
public String toString(){
    if(items==1)
        return "List of "+items+" item";
    else
        return "List of "+items+" items";
}
```

```
customerList List of 1 item
furntiures List of 3 items
```

Assignment 2.2

Friday, February 3, 2023 11:06 AM

- Define `toString` in `List` that should display the list items

```
List customerList=new List();

customerList.add("Vivek");
customerList.add("Sanjay");

List furntiures=new List();

System.out.println(customerList);
System.out.println(furnitures);
```

[Expected Output](#)

[
 \tVivek\tSanjay\t]

(empty)

Non Initialized variables

Friday, February 3, 2023 11:17 AM

- We have two types of variables we declare in Java
 - Fields
 - we declare inside the class
 - They represent the property of an object
 - They are by default initialized to
 - null for objects
 - 0 for numbers
 - false for boolean
 - " for char
 - Method local variables
 - They remain uninitialized till you initialize them
 - You can't use them without first initializing them

```
class Program{  
    int number; //by default 0  
    String name; //by default null;  
  
    void f1(){  
        int x; //un-initialized  
        String y; //uninitialized  
  
        y="Hi";  
  
        System.out.println(y); //works  
  
        System.out.println(x); //fails. not  
        initialized  
  
        System.out.println(number); //works 0  
  
        System.out.println(name); //works null  
    }  
}
```

Organizing Large Application

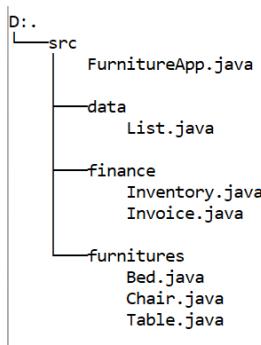
Friday, February 3, 2023 11:32 AM

- We have multiple classes in our code
 - These classes represent different domain elements
 - Furnitures
 - Chair
 - Table
 - Bed
 - Generic Data
 - List
 - Finance
 - Invoice
 - Inventory
-
- We don't want to keep all our files at one place
 - Generally we may not want to include our source file in distribution
 - How do we organize
 - Source and class files separately
 - Each domain related classes separately

Folder Based Organization

- we can create separate folder for
 - src
 - should contain .java files
 - class
 - should contain .class files
- Inside both these folders we can have sub folders representing domains
 - furntiures
 - data
 - finance

Our Project src structure



Will the compilation work?

```
C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>javac FurnitureApp.java
FurnitureApp.java:6: error: cannot find symbol
      List customerList=new List();
           ^
symbol:   class List
location: class FurnitureApp
FurnitureApp.java:6: error: cannot find symbol
      List customerList=new List();
           ^
symbol:   class List
location: class FurnitureApp
FurnitureApp.java:10: error: cannot find symbol
      List furnitures=new List();
```

Problem

- Java doesn't know where to go looking for classes (source or bytecode)

Solution : CLASSPATH

- Just like environment variable PATH that helps us locate executable files (eg. java, javac), java uses a system environment variable CLASSPATH to look for all the paths where classes are likely to be present
- A class path can maintain a list of PATH separated by
 - ; in windows OS
 - : in linux and MAC
 - These are as per the OS convention
- To make our design work we need to include all folders where we expect the path

Setting class path

- There are multiple ways to set the CLASSPATH

Option#1 In the environement setting of your system



```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>echo %classpath%
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src\furnitures;D:\MyWorks\Corporate\202302-anz-java\java-demos\de
mo04\src\data;D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src\finance

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>javac FurnitureApp.java
```

```
volume serial number is F8C2-FD66
:.
└── src
    ├── FurnitureApp.class
    └── FurnitureApp.java

    ├── data
    │   ├── List.class
    │   └── List.java

    ├── finance
    │   ├── Inventory.class
    │   ├── Inventory.java
    │   ├── Invoice.class
    │   └── Invoice.java

    └── furnitures
        ├── Bed.class
        ├── Bed.java
        ├── Chair.class
        ├── Chair.java
        └── Table.class
            └── Table.java
```

A Small Snag

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>dir
Volume in drive D is Data
Volume Serial Number is F8C2-FD66

Directory of D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src

02/03/2023  11:47 AM    <DIR>      .
02/03/2023  11:36 AM    <DIR>      ..
02/03/2023  11:47 AM    <DIR>      data
02/03/2023  11:47 AM    <DIR>      finance
02/03/2023  11:47 AM    <DIR>      furnitures
```

Why are we unable to find class file present in the current directory?

- By default Java/javac searches for class (both source/butcode) in the current working directory

```

Directory of D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src

02/03/2023 11:47 AM <DIR> .
02/03/2023 11:36 AM <DIR> ..
02/03/2023 11:47 AM <DIR> data
02/03/2023 11:47 AM <DIR> finance
02/03/2023 11:47 AM 1,783 FurnitureApp.class
02/03/2023 11:31 AM 1,092 FurnitureApp.java
02/03/2023 11:47 AM <DIR> furnitures
      2 File(s)     2,875 bytes
      5 Dir(s) 141,212,073,984 bytes free

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>java FurnitureApp
Error: Could not find or load main class FurnitureApp
Caused by: java.lang.ClassNotFoundException: FurnitureApp

```

Current Directory :

- By default Java/javac searches for class (both source bytecode) in the current working directory
- Once CLASSPATH is set it searches only in the directories mentioned in the classpath
 - they stop searching in current folder

Solution

- We can always ask java to search in current directory by adding "." path in the CLASSPATH

Aside

- appending value to existing environment variable like class path

Windows

```
set CLASSPATH = %CLASSPATH%;.\something
```

Linux/Mac

```
set CLASSPATH = $CLASSPATH:./something
```

Why we shouldn't set classpath at the system level?

- we may have many applications
- Each will need its own classpath
 - they may conflict with each other

Option#2 creating classpath at application level

- we can declare the classpath directly on the command window
- Problem
 - we need to set the classpath everytime we open a command window
- Solution
 - Use a batch file / shell script

Option #3 setting the classpath directly on java/javac using -cp switch

```

C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04>java FurnitureApp
Error: Could not find or load main class FurnitureApp
Caused by: java.lang.ClassNotFoundException: FurnitureApp

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04>java -cp .\src;.src\furnitures;.src\data;.src\finance FurnitureApp
Total Customers: 1
Furnitures: 3
c1.price=3000  c2.price=2000
Table@776ec8df
Bed@4eec7777
Inventory@3b07d329
Invoice@41629346
Invoice@41629346
Invoice@404b9385
customerList [ Vivek ]
furnitures [ Chair Table Bed ]
orders (empty)

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04>echo %CLASSPATH%
%CLASSPATH%

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04>

```

No classpath set

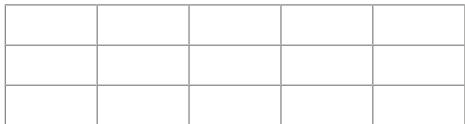
CLASSPATH is set using -cp switch

How to separate source and class files in different folders

Class name conflict

Friday, February 3, 2023 12:27 PM

What is a Table?



Data Table

Or

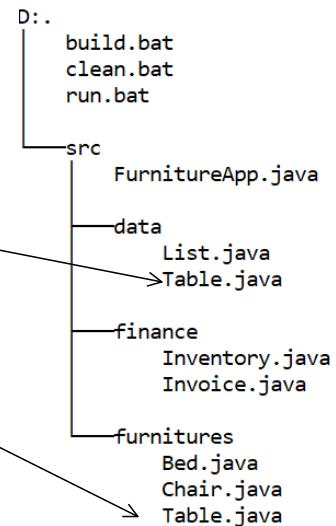


Furniture Table

Do we ever need both the table in the same application?

Which Table is referred here?

```
Table t1=new Table();
System.out.println(t1);
```



How will java decide which table to use?

- Java reads the class path in sequence in which it defined
- In our example we include "data" folder before "furnitures" folder

```
set APP_ROOT=.
set SRC=%APP_ROOT%\src
set CLASSES=%APP_ROOT%\classes
set CP=%CLASSES%\data;%CLASSES%\finance;%CLASSES%\furnitures
javac -d %CLASSES%\data %SRC%\data\*.java
javac -d %CLASSES%\finance %SRC%\finance\*.java
javac -d %CLASSES%\furnitures %SRC%\furnitures\*.java
javac -d %CLASSES% -cp %CP% %SRC%\FurnitureApp.java
```

- As such it will be using data table and NOT furniture table
- A classpath search stops the moment a candidate class is located.

How to use both Table in the same class

- we can't keep them in same folder
- classpath will check the first folder only

Java Packages

Friday, February 3, 2023 12:39 PM

- Java Package is a logical grouping of classes and (sub) packages
- We can create packages like

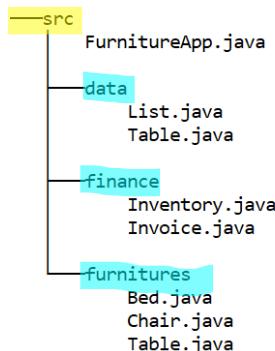
- **furnitures**
 - Chair
 - Table
 - Bed
- **data**
 - List
 - Table
- **finance**
 - Invoice
 - Inventory

What is the difference between a package and folder

- A package is a "java" concept, folder is an "os" feature
- A package name will be used within the java program, folders appear only externally in classpath
- A package will still be using folder structure
 - A package is not a folder
 - A package lives inside a folder

How do we create a package

- we can designate any folder as a package
 - we may designate a nested folder structure as nested package
- The package appears in the source code

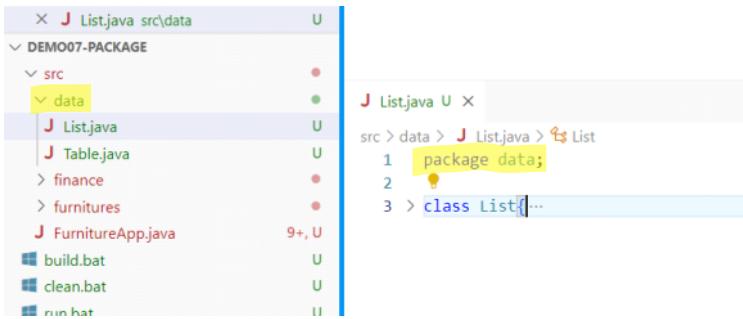


- Here we have designated following folder as packages
 - data
 - finance
 - furnitures
- A package becomes language concept and doesn't appear in classpath
- we are not considering "src" as package
 - src is a container path for packages
 - This folder will appear in the CLASSPATH
- Why is "src" not a package?
 - Because we don't want

Step 1 Designating Package for Class



- package "data" should match the immediate folder structure
 - more important for .class file than .java file
- if we chose to name our package as src.data
 - src package will contain subpackage data
- In our example data is package, src is container path for the package.



- If we chose to name our package as src.data
 - src package will contain subpackage data
- In our example data is package, src is container path for the package.

Compiling Classes from a Package

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>javac -d .\classes .\src\data\*.java
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>tree/
Folder PATH listing for volume Data
Volume serial number is F8C2-FD66
D:.
    build.bat
    clean.bat
    run.bat

    classes
        data
            List.class
            Table.class
```

- even when we have marked to store compiled classes to "classes" folder because they belong to a package "data" compiler generates the package folder and stores it

Step 2 Using classes from a Package

- Now we don't have a non-packaged (global) class like
 - Chair
 - Table
 - List
- We have classes like
 - furnitures.Chair
 - furnitures.Table
 - data.Table

Note

- Now we have two distinctly identifiable Tables
 - furnitures.Table
 - data.Table

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>javac -cp .\classes\data -d .\classes .\src\FurnitureApp.java
.\src\FurnitureApp.java:6: error: cannot access List
        List customerList=new List();
                           ^
bad class file: .\classes\data\List.class
  class file contains wrong class: data.List
  Please remove or make sure it appears in the correct subdirectory of the classpath.
1 error
```

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>

Problem

- classpath includes "data" as sub folder
- compiler enters this folder and tries to search for package "data" which is not present

Solution

- package name shouldn't be part of classpath
- parent of package should be part of classpath
 - classpath is used for searching both
 - class
 - package

Problem 2

```
C:\Windows\System32\cmd.exe

D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>javac -cp .\classes -d .\classes .\src\FurnitureApp.java
.\src\FurnitureApp.java:6: error: cannot find symbol
    List customerList=new List();
           ^
symbol:   class List
location: class FurnitureApp
.\src\FurnitureApp.java:6: error: cannot find symbol
    List customerList=new List();
           ^

```

- Now It is searching for List class in classes folder
 - It doesn't exist
- In fact we don't have a global List class
 - we have data.List

Solution 2.1 (Step 2.1) using package qualified names

Problem #3

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>javac -cp .\classes -d .\classes .\src\FurnitureApp.java
.\src\FurnitureApp.java:6: error: List is not public in data; cannot be accessed from outside package
    data.List customerList=new data.List();
           ^
.\src\FurnitureApp.java:6: error: List is not public in data; cannot be accessed from outside package
    data.List customerList=new data.List();
           ^
.\src\FurnitureApp.java:10: error: List is not public in data; cannot be accessed from outside package
    data.List furnitures=new data.List();
           ^
.\src\FurnitureApp.java:10: error: List is not public in data; cannot be accessed from outside package
    data.List furnitures=new data.List();
```

- So far all our classes belonged to an unnamed global package
- They all belonged to same family and can access each other without problem
- Now List belongs to a different package "data" and can't be accessed outside the package unless it is marked public
 - same goes true for list members
 - add
 - remove
 - count

```
public static void main(String []args){
    data.List customerList=new data.List();
    customerList.add(item: "Vivek");

    data.List furnitures=new data.List();
    furnitures.Table t1=new furnitures.Table();
    data.Table t2=new data.Table();
```

Advantage

- We can access both Tables
 - data.Table
 - furnitures.Table
- We have smaller class paths
 - we don't need packages folders to be part of classpath
- Easy compile and run
 - we just need one class path
 - It can compile all dependency classes properly
- Auto organization of classes in right package folders

Problem

- We need to include the package qualified name everywhere
- When we have many classes (we always have many classes) package qualified names becomes difficult

Option 2.2 import statement

- we can import a particular package contents (classes) directly so that we can use them without qualifying the package name

```
💡  
import finance.*; //get all the classes from finance package  
  
class FurnitureApp{  
  
    public static void main(String []args){  
  
        Invoice i1=new Invoice();  
        Invoice i2=new Invoice();  
        Inventory inventory=new Inventory();  
  
        System.out.println(i1);  
        System.out.println(i2);  
        System.out.println(inventory);  
    }  
}
```

Note

- import "*" can import all classes from a package not the sub packages
- there is nothing like ".*"
- Once imported you can use all the classes from there

Problem with * import

- We generally avoid importing the entire package
- If we import all package with "*" it will be problem similar to not having package

```
Table t1=new Table();  
  
C:\Windows\System32\cmd.exe  
  
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo07-package>build  
.src\FurnitureApp.java:20: error: reference to Table is ambiguous  
    Table t1=new Table();  
           ^  
  both class furnitures.Table in furnitures and class data.Table in data match  
.src\FurnitureApp.java:20: error: reference to Table is ambiguous  
    Table t1=new Table();  
           ^  
  both class furnitures.Table in furnitures and class data.Table in data match  
2 errors
```

Option #3 importing a class selectively from a package

- you may specify which class you want to import

```
import finance.*; //get all the classes from finance package
import data.*;
import furnitures.*;
import furnitures.Table;
```

```
class FurnitureApp{

    public static void main(String []args){

        Invoice i1=new Invoice();
        Invoice i2=new Invoice();
        Inventory inventory=new Inventory();

        List customerList=new List();
        List furnitures=new List();

        Table t1=new Table();
```

What if we need both Tables

- In such cases we need to use one of the reference explicitly as fully qualified name

```
import furnitures.Table;

class FurnitureApp{

    public static void main(String []args){

        Invoice i1=new Invoice();
        Invoice i2=new Invoice();
        Inventory inventory=new Inventory();

        List customerList=new List();
        List furnitures=new List();

        Table t1=new Table(); //furnitures.Table

        data.Table t2=new data.Table(); //explicit selection
```

Recommendation

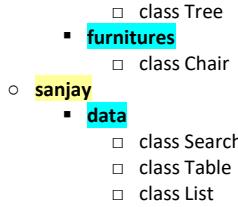
- Java best practice guidelines recommends importing classes rather than package.*

What is the possibility that two different programmer will create a package with same name and have same class inside it

- High possibility
- Package is a bundle of related classes
- If package name is same chances are we will create classes also in the same way

What is the possibility that we need packages created by two developers in the same project

- **src**
 - class App
 - **vivek**
 - **data**
 - class List
- **folder**
- **package**



How do I access both Tree and Search class?

- we can have both src\anjay and src\vivek in CLASSPATH
- Now we have a **single package (logical entity) called data** which holds
 - List
 - Tree
 - Search
 - Table
 - List

- When we see multiple package definitions they merge as a single logical unit

compilation

```
$ javac -d .\classes -cp .\src\vivek;.\src\anjay;.\src App.java
```

Run

```
$ java -cp .\classes App
```

What if I want to access List?

- Here it will access the List from that package folder which appears first in CLASSPATH

How do I access both the List?

- There is NO Java WAY.

Takeaway

- Contents inside two classes never conflict
 - class acts as boundary
 - Two classes can have field and methods with same name
- Class names may conflict
- This conflict can be resolved using packages
- Package names don't conflict. They merge
- When we have two package folders (same package) with same class, then we have a problem that can't be resolved
 - whichever folder is first in the list will be used.
 - other is unreachable

Solution

- to avoid conflict within the package, we use the concept of nested package
- generally we use outer package as identity space (identification of the creator)
 - Example
 - package vivek.data
 - package sanjay.data
- **What is the possibility of name conflict between vivek.data and sanjay.data**
 - vivek and sanjay are very common names and most likely will conflict

Package naming convention

- A package should always be nested
- The outer most package (generally 2) defines identity
 - conventionally we use reverse domain as unique identity

- in.conceptarchitect
 - org.apache
 - com.anz
 - starting third level package we may use for logical grouping
-
- example
 - in.conceptarchitect.commons.data
 - in.conceptarchitect.commons.finance
 - in.conceptarchitect.app.furnitureapp.furnitures
 - in.co.sanjay.data

Distribution

Friday, February 3, 2023 3:01 PM

- A java project will have typically hundreds of classes in dozens of packages and sub packages (folders and sub folders)yste
- Distributing files this way is goint to be difficult
- Java provides a simpler alternative

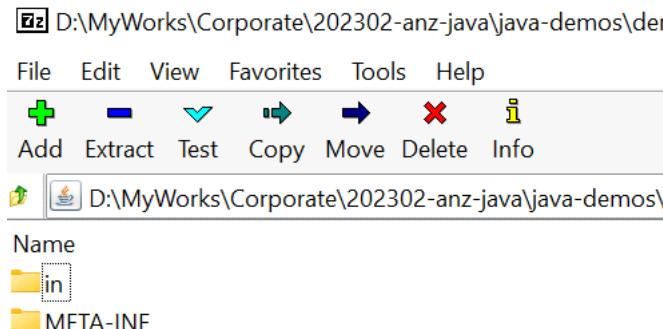
Jar file

- Jar stands fro Java Archive
- Concept is similar to a zip file
- You bundle the class and package in a single file
- Java can run the application without uncompressing this file
- It improves performance as you will have fewer I/O tor ead the archive

To create a Jar file

```
jar -cf ..\app.jar .
```

- create a jar file including all files and folder and sub folder from current folder
- the jar file should be saved as ..\app.jar



- A jar contains all my files
- It also include some Meta information needed by Java

Running the program from Jar

- we can just use the jar file as class path

Manifest

- can include any information realated to jar as key value pair
- we need to create our own manifest file and add the information
- information provided by us shall be merged in actual manifest

Recursive Function

Monday, February 6, 2023 9:19 AM

- A Recursive Function is a Function Calling Itself
- Sometimes a large and complex algorithm can be broken up into some other term of itself
- Example
- Factorial of 5

5 x 4 x 3 x 2 x 1

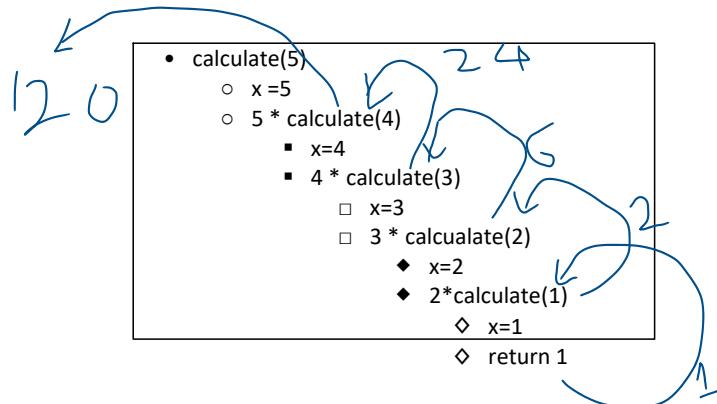
Factorial of 4

- Now we can say

$$5! = 5 * 4!$$

- Programmatically we can write

```
Factorial.java
package in.concpetarchitect.maths;
public class Factorial {
    int calculate(int x) {
        if(x<=1)
            return 1;
        else
            return x* calculate(x-1);
    }
}
```



How many x we have?

- There are 5 different "x" variable present in memory at this point in time
 - each x is different from each other
- a new set of variables are created each time a method is called
 - a new set of parameters
 - a new set of all local variables declared within the method
- A class level field (static) is created only once
- An object level (non-static) field is created per object

Important Note

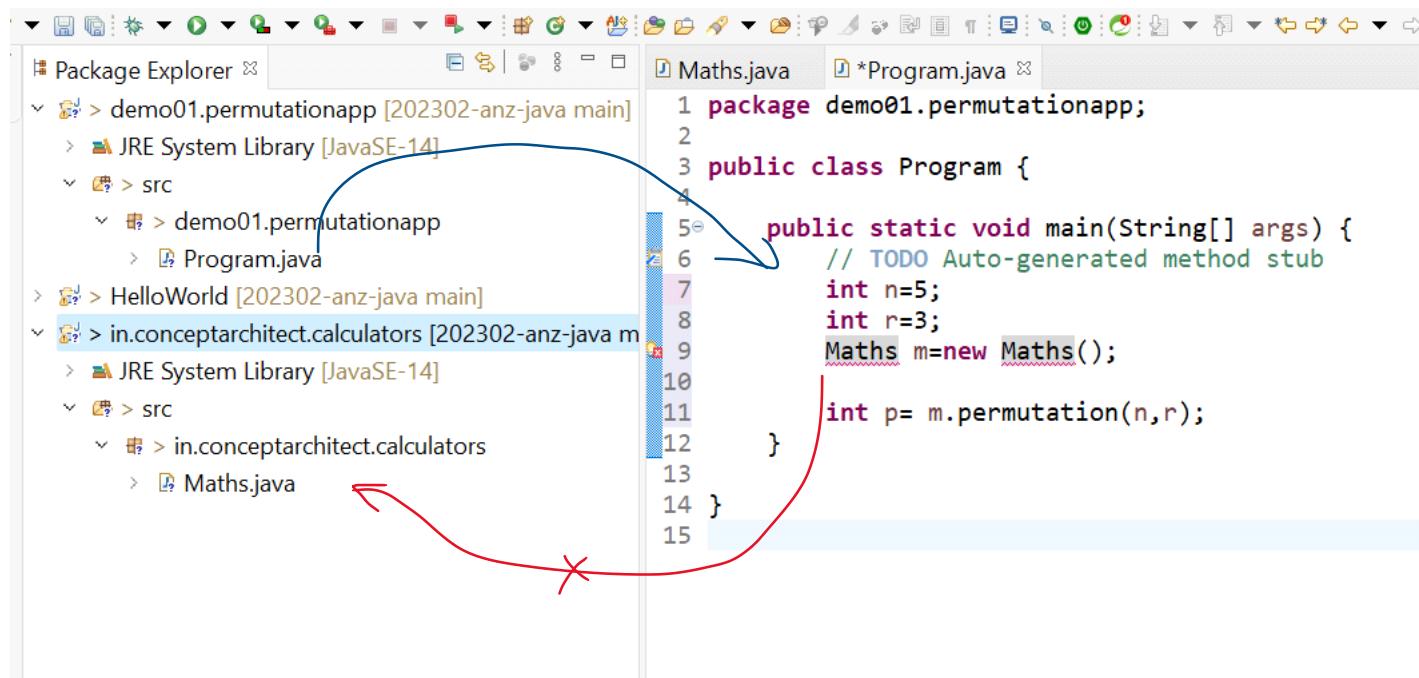
- for every recursive function there should be at least two returns
- One recursive return (that is why it is recursive function)
- One non-recursive direct return
 - If we don't have a direct return the method may enter infinite loop causing "stack overflow error"

Using class from other eclipse Project

Monday, February 6, 2023 10:24 AM

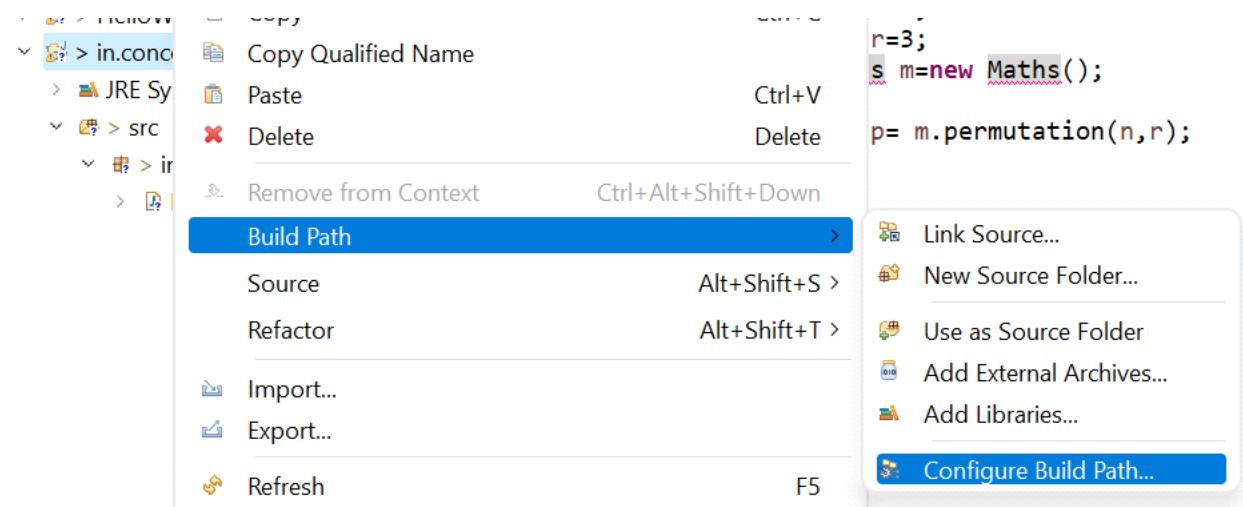
How do I access class from other projects

- By default eclipse will search the classes within the project itself
- ctrl+space or ctrl+shift+o will not get details from other projects within or outside the workspace

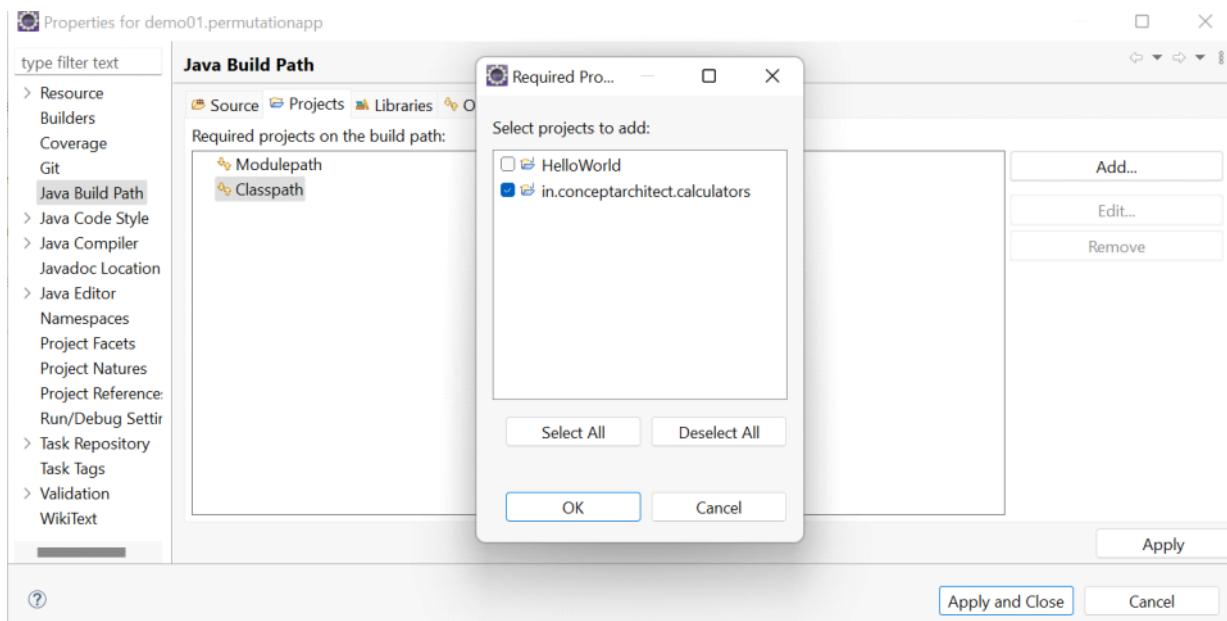


Adding Reference of One Project in another

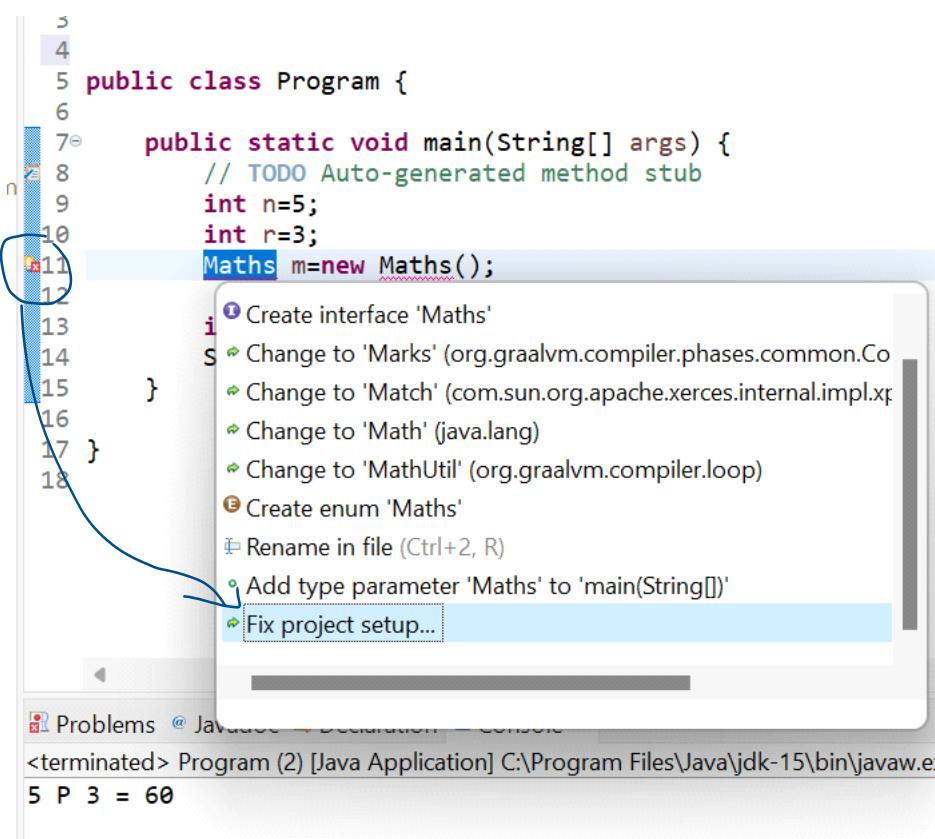
1. Right click project and select the option



2. Select the Project in the build-path



Alternative Option





Project Setup Fixes



The following proposals have been found to fix the unresolvable reference to 'Maths':

- Add project 'in.conceptarchitect.calculators' to build path of 'demo01.permutationapp'

Click [here](#) to manually configure the build path of project 'demo01.permutationapp'.



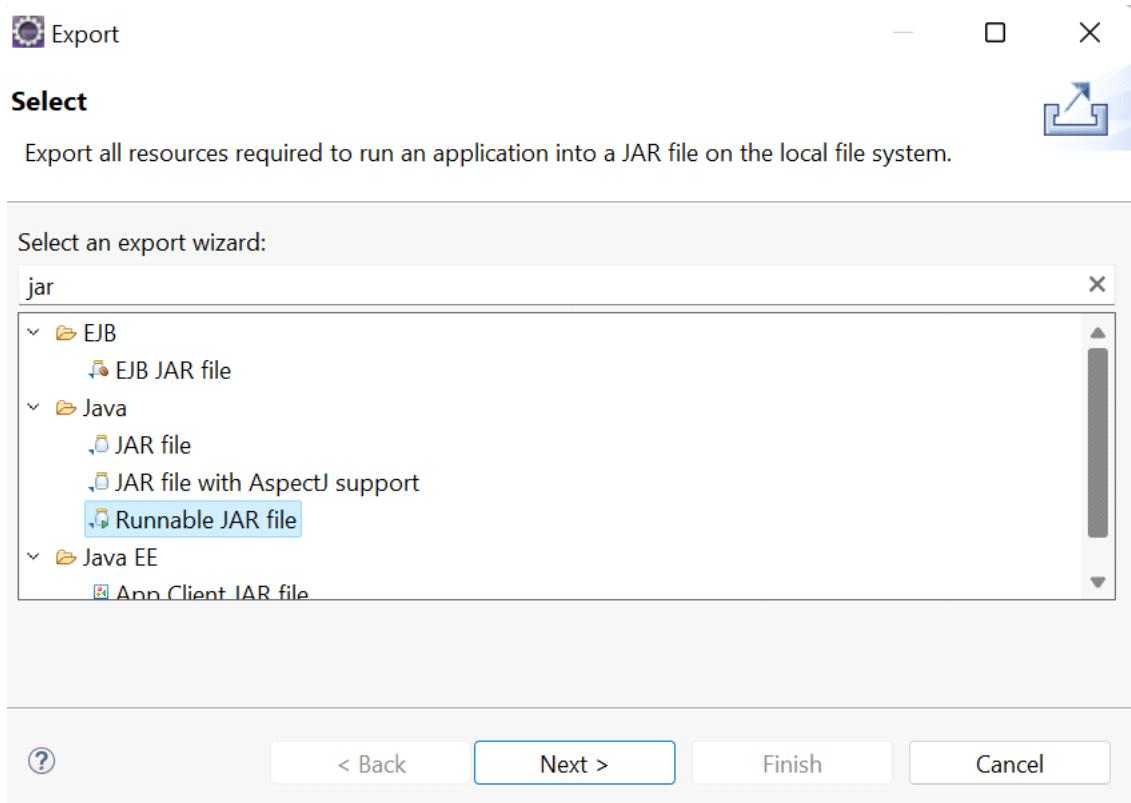
OK

Cancel

Creating jar from eclipse

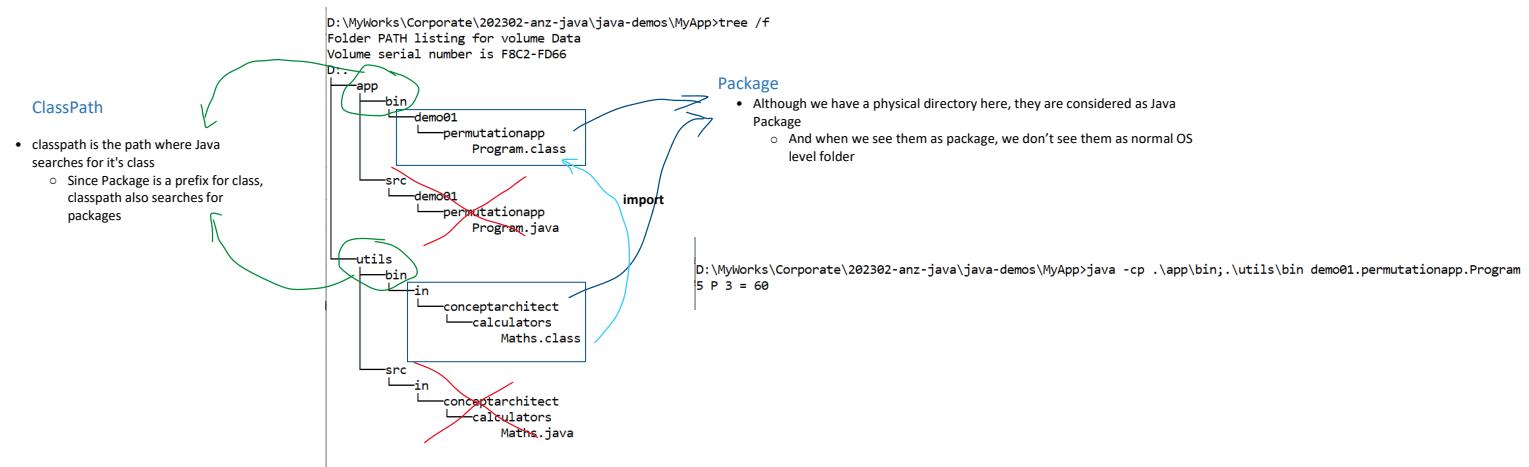
Monday, February 6, 2023 10:36 AM

1. File → export



Understanding Runtime Classpath

Monday, February 6, 2023 10:55 AM



Tuesday, March 7, 2023 8:18 AM

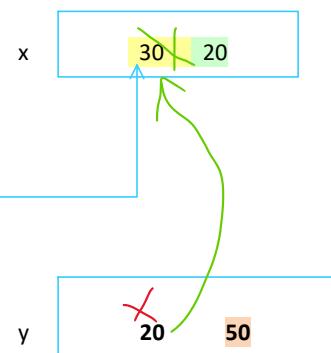
```
class Bank{  
  
    public void Transfer(BankAccount a1, String password, double amount, BankAccount a2 ){  
  
        a1.withdraw(amount, password);  
        a2.deposit(amount);  
    }  
}
```

Primitive Type (Value type) Memory allocation

Monday, February 6, 2023 11:38 AM

```
int x; //memory is allocated for 1 int. It may not be initialized yet
```

```
x = 30;
```



```
int y=20; //memory allocated and initialized together
```

Assigning another value to a variable

```
x=y;
```

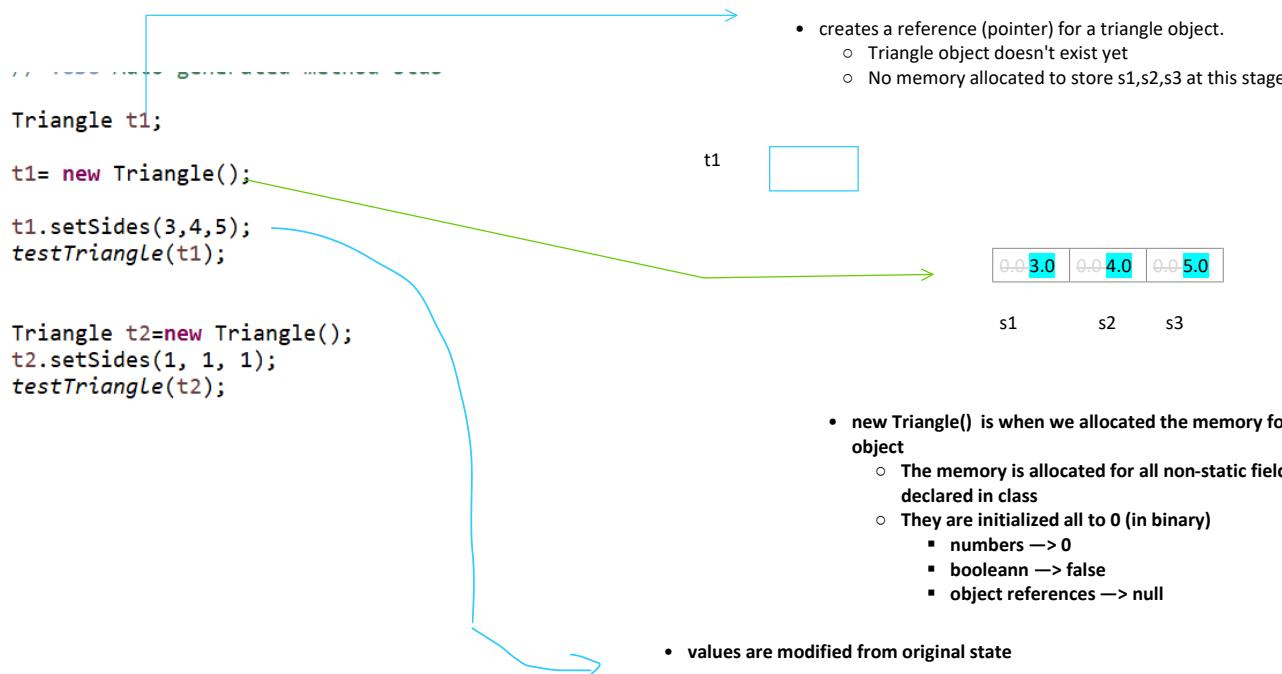
- current value of y is copied to X. But they are not related any further

```
y=50;
```

- replaces the value of y from 20 to 50
- No change in the value of x, which is unrelated

Object (Ref type) Memory Allocation

Monday, February 6, 2023 11:35 AM



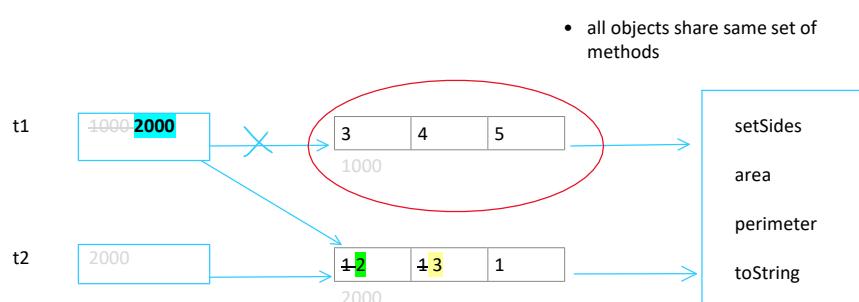
IMPORTANT NOTE

- Memory is only allocated for non-static fields declared inside the class
- No memory is allocated at this stage for
 - Methods of the class
 - A common copy is used for all objects
 - Any method parameter or method local
 - They are allocated when you call the method
 - Any class field marked static
 - A single copy is stored in memory
 - More on this later.

Working with multiple Objects

```
Triangle t1= new Triangle();
t1.setSides(3,4,5);
```

```
Triangle t2=new Triangle();
t2.setSides(1,1,1);
```



Assigning One Object to another

```
t1=t2;
```

- Here the reference to `t2` will be copied to `t1`
 - It will not copy the object contents
 - Just the reference
- At this stage both `t1` and `t2` are referring to same object
 - `Triangle<1,1,1>`

What happens to Object `Triangle<3,4,5>`?

- In self managed codes (like c/c++) this is considered as a memory leak
 - This object has no reference and it can't be used.
 - It will remain in memory forever (till the app is running)
 - Their memory can't be re-used
 - We MUST free the memory by deleting the object before this type of assignment

- It will not copy the object contents
- Just the reference
- At this stage both t1 and t2 are referring to same object
 - Triangle<1,1,1>

If we change any one, it changes both

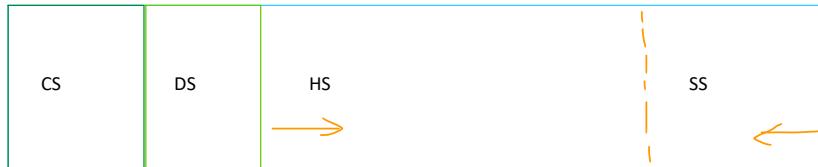
```
t1.s1=2;
t2.s2=3;
```

Actually we have just one object here which is referred by two different references

- Any change by any reference changes the same object.
- both t1 and t2 will have the same hashcode as they are the same object

- It will remain in memory forever (till the app is running)
- They memory can't be re-used
- We MUST free the memory by deleting the object before this type of assignment
- In Managed languages like (Java/C++/Python/JavaScript/...)
 - The runtime has a process called "garbage collection"
 - The process frees memory at un undeterministic interval
 - All the un-referenced memory may be freed by garbage collector by its own strategy or convinience
 - It is a complex and evolving process
 - It involves several generations
 - garbage collection typically works when
 - we start to run out of memory
 - when system resources are comparatively free/idle
 - Even when garbage collector runs there is no surity that it will free all the memory.
 - It may free just one generation of memory
 - Java has a API to force garbage collection
 - This api can initiate garbage collection
 - Even this api doesn't guarantee IMMEDIATE
 - ◆ It is meant to be suggestive not authoritative
 - ◆ although generally it is IMMEDIATE
 - In MOST cases it is not recommended to interfere in the process.

Memory Layout of a typically Application



- Code Segment
 - stores your logic
 - class methods
- Data Segment
 - Stores static and const values
- Heap Segment
 - Stores dynamically allocated memory
 - "new"
- Stack Segment
 - Stores method locals and parameter
- There is no hard division between Stack and Heap
 - They grow towards each other dynamically

Java Heap Management

- Java Heap Management works on assumptions that
 - An object either dies very young or lives to grow old.
 - there will be many objects created within a method call and are not used once the call is over.
 - Few objects are required throughout application and may live for entire life
- Java Garbage collection has generation model
 - Gen 0
 - Gen 1
 - Gen 2
 - ...
- All objects are always created on Gen1 Heap
 - few will be dead long before garbage collection
 - others may live on.
- When garbage collectors starts it starts for a particular generation (not for everyone)
 - Gen1 garbage collector may start when gen0 heap is (almost) full
 - It checks for all living objects (not dead ones)
 - It moves all the surviving objects to gen 1
 - Note address will change and that is why Java never gives the address to program
 - All the references to this object are automatically changed to new address

- All the dead objects are removed and gen 0 is now completely empty
- Same thing will happen to gen1 and gen2

Triangle Revisited

Monday, February 6, 2023 1:23 PM

```
Triangle t3=new Triangle();
t3.setSides(2,4,8);
testTriangle(t3);
```

- As per java we have a valid object referred by t3.
- But we can't create a triangle with dimension 2,4,8
- Geometrically (domain rule) for a valid triangle
 - sum of every two sides must be greater than the third.

How do I model a valid Triangle?

- we need to incorporate the triangle rule in the domain object.

Approach #1

- validate value before assigning and display error message otherwise

```
3 public class Triangle {
4
5     double s1,s2,s3;
6
7     void setSides(double x, double y, double z) {
8         if(x>0 && y>0 && z>0 && x+y>z && y+z>x && x+z>y) {
9             s1=x;
10            s2=y;
11            s3=z;
12        } else {
13            //what to do when user enters wrong info?
14            System.out.println("invalid sides");
15        }
16    }
17
18
19    double perimeter() {
20        return s1+s2+s3;
21    }
22}
```

- indicates we have error

Error display may not always be best option

- This display doesn't prevent perimeter calculation.
- Any value returned will essentially be wrong as invalid triangle shouldn't have a perimeter.
- perimeter function has no knowledge of any message displayed by setSides.

Approach #2 set a flag (indicator) to mark triangle valid or invalid

```
4
5 public class Triangle {
6
7     double s1,s2,s3;
8     boolean valid;
9
10    void setSides(double x, double y, double z) {
11        if(x>0 && y>0 && z>0 && x+y>z && y+z>x && x+z>y) {
12            s1=x;
13            s2=y;
14            s3=z;
15            valid=true;
16        } else {
17            //what to do when user enters wrong info?
18            //System.out.println("invalid sides");
19            valid=false;
20        }
21
22    double perimeter() {
23        if(valid)
24            return s1+s2+s3;
25        else
26            return Double.NaN;
27    }
28}
```

- Here we have a Triangle Object
- When we setSides it also internally sets a valid flag to specify if triangle is valid or not
- other behavior of this triangle respects "valid" status and returns expected answers for valid and invalid scenario.

Binding of Data and Behavior (Encapsulation)

- In this object the triangle states (s1,s2,s3,valid) are interconnected.
- setSide sets the values as per the requirement
- area() and perimeter() are also connected to the same triangle rule and represents proper domain model
- internally all the states and behavior together represent Triangle
- Encapsulation is more about defining a responsibility.**

But what if client is not reasonable/responsible?

```
//t1 is a valid triangle  
//What if I change one of it's side making it invalid  
t1.s1=100; //the valid flag is not changed.  
  
testTriangle(t1);
```

- we may bypass setSides and change values directly
- Now s1 has changed but valid flag is not reset

```
Triangle <100.0,4.0,5.0>  
Perimeter: 109.0 Area: NaN
```

```
//t3 is invalid (sides are 0,0,0)  
//but we can change the valid field  
  
t3.valid=true; //a simple lie.  
testTriangle(t3);
```

```
Triangle <0.0,0.0,0.0>  
Perimeter: 0.0 Area: 0.0
```

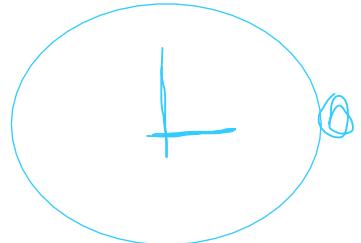
- here Triangle was marked invalid by setSide
- But we can unmark it by resetting valid flag from outside the Triangle object

Encapsulation recommends protection against unwanted changes.

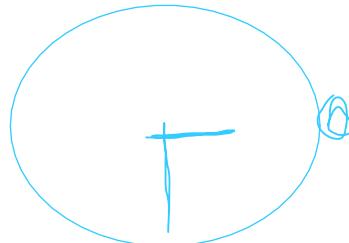
- In our code we should not allow anyone to access my state (data) directly
- It should change using authorized behavior model.

Scope Rules

- public
 - → accessible by everyone
- (package)/no scope
 - → accessible by everyone within the same package
- private
 - accessible only within the class and not outside
- protected
 - to be discussed later.



Why do we need a time changing knob and not manually shift hand?



- Ideally when min hand moves the hour hand too should move

```
//t1 is a valid triangle  
//What if I change one of it's side making it invalid  
t1.s1=100; //the valid flag is not changed.  
  
testTriangle(t1);  
  
//t3 is invalid (sides are 0,0,0)  
//but we can change the valid field  
  
t3.valid=true; //a simple lie.  
testTriangle(t3);
```

- with private s1 and valid no body can tamper with this information
- This information will be set only using right set of values

Problem → what if we need to see the value

```

private static void testTriangle(Triangle triangle) {
    System.out.println(triangle);
    if(triangle.valid) {
        System.out.print("Perimeter: "+triangle.perimeter());
        System.out.println("Area: "+triangle.area());
    }
    System.out.println();
}

```

- Now outsiders can't even see if triangle is valid or not
- It is a valid use case.

Solution —> define a method to return the valid/invalid status

- we should allow only checking for the information and not changing it.

```

private double s1,s2,s3;
private boolean valid;

public boolean isValid() {
    return valid;
}

```

- Here we can check the validity but not change from outside

A little refactor

- let's change the valid flag to invalid flag

Wh

Problem — What if we never call setSides?

```

27
28     Triangle t4=new Triangle();
29
30     testTriangle(t4);
31

```

- triangle by default is "valid"
- if no side is set it may consider side 0 to be valid

When is the triangle created : Line 12 or Line 14?

```

11
12     t1= new Triangle();
13
14     t1.setSides(3,4,5);
15
16
17     testTriangle(t1);

```

- If Triangle is created on Line 12
 - what are the sides of triangle on line 13?
 - can a "valid" triangle exist with side 0?
- If Triangle is created on Line 14
 - What was the value of "t1" on line 13?
 - what is the value of t1.perimeter() on line 13?

Right answer

- There are two creations here
- Java Object is created on Line 12
 - memory is allocated
 - but the object is not a geometrical triangle
 - it is not usable yet.
- Geometrical triangle is not ready till line 14 is called
 - This is where domain object is created
- Real Problem
 - There are two ideas
 - java object
 - domain object
 - Creations are not in sync
 - there is a gap

Object Oriented Concept → Constructor

- constructor is a special class method with same name as that of class

```
Triangle t= new Triangle();  
|-----> constructor of the class object  
|  
|-----> class name
```

- The two are same name but different concepts

- This method has no return type but returns the newly created object
- This method is called for creating the object with new
- Every class contains a default nothing doing constructor
- We can define our own constructor that replaced the default one.
 - if we define a constructor the default will be removed.
- Our constructor can take multiple arguments
 - We can have overloaded constructor
 - multiple constructor taking different number or type of parameter

Program.java:

```
7 public static void main(String[] args) {  
8     // TODO Auto-generated method stub  
9  
10    Triangle t1;  
11  
12    t1 = new Triangle();  
13    t1.setSides(1, 1, 1);  
14    testTriangle(t1);  
15  
16    Triangle t2;  
17    t2.setSides(1, 1, 1);  
18    testTriangle(t2);  
19  
20    Triangle t3=new Triangle();  
21    t3.setSides(2,4,8);  
22    testTriangle(t3);  
23  
24  
25
```

Triangle.java:

```
1 package in.conceptarchitect.shapes;  
2  
3 public class Triangle {  
4  
5     private double s1,s2,s3;  
6     private boolean inValid;  
7  
8     public boolean isValid() {  
9         return !inValid;  
10    }  
11  
12    public Triangle(double s1,double s2,double s3) {  
13        setSides(s1,s2,s3);  
14    }  
15  
16    public void setSides(double x, double y, double z) {  
17        if(x>0 && y>0 && z>0 && x+y>z && y+z>x && x+z>y) {  
18            s1=x;  
19            s2=y;  
20            s3=z;  
21            inValid=false;
```

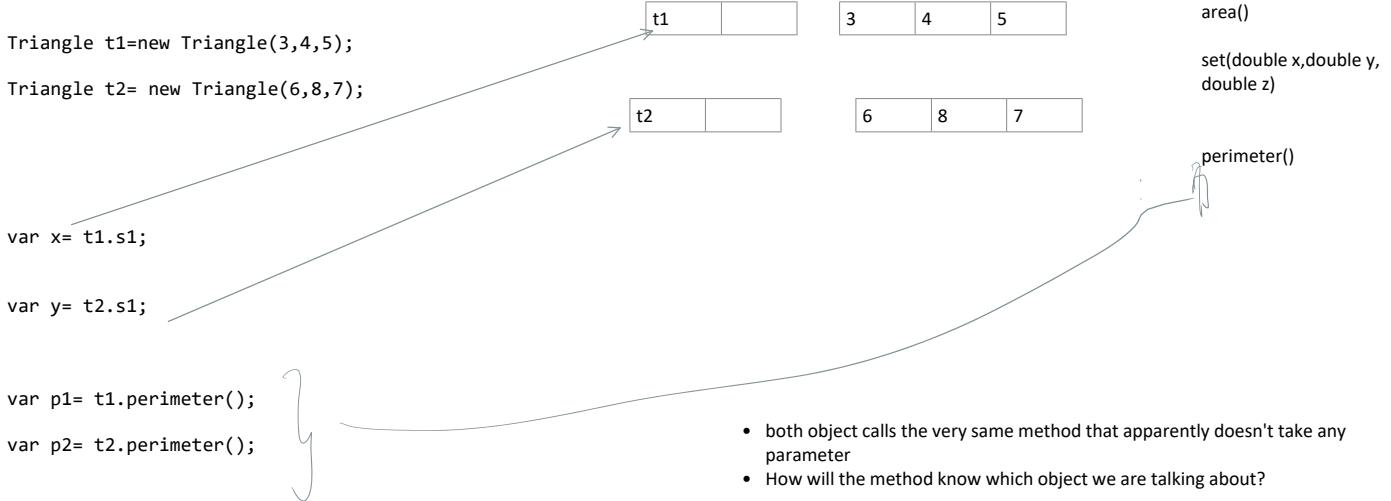
Assignment 3.1

Monday, February 6, 2023 2:24 PM

- Create a model for a Bank Account
- We should have following information
 - name
 - account number
 - balance
 - interest rate
 - password
- It should support following operations
 - deposit
 - should reject negative amount
 - withdraw
 - should fail for
 - negative amount
 - access withdrawal
 - invalid password
 - credit interest
 - gives one month interest with formula
 - $\text{balance} = \text{balance} * \text{rate} / 1200$
 - to string
 - to show the account object as string
- Write a test app to work with bank account

'this'

Tuesday, February 7, 2023 8:25 AM



'this' reference

- every object method gets an additional parameter 'this' when invoked
- this always refers to the invoking object
- You can conceptually understand that all our object methods are actually like a global method and are slightly modified by the compiler (conceptual view)

//conceptual view

```
class Triangle{
    double s1,s2,s3;
    public double perimeter(){
        return s1+s2+s3;
    }
}

class Program{
    public static void main(String []args){
        Triangle t1=new Triangle(3,4,5);
        Triangle t2= new Triangle(4,4,4);
        var p1=t1.perimeter();
        var p2=t2.perimeter();
    }
}
```

class Triangle{
 double s1,s2,s3;
 public static double perimeter(Triangle this){
 return this.s1+this.s2+this.s3;
 }
}

```
class Program{
    public static void main(String []args){
        Triangle t1=new Triangle(3,4,5);
        Triangle t2= new Triangle(4,4,4);
        var p1=Triangle.perimeter(t1); //t1.perimeter();
        var p2=Triangle.perimeter(t2); //t2.perimeter();
    }
}
```

Banking Model

Tuesday, February 7, 2023 8:51 AM

```
public void withdraw(double amount, String password) {  
    if(amount<0)  
        System.out.println("withdraw failed for negative amount");  
    else if(amount>balance)  
        System.out.println("insufficient balance");  
    else if(!this.password.equals(password))  
        System.out.println("invalid credentials");  
    else {  
        balance-=amount;  
        System.out.println("Please collect your cash");  
    }  
}
```



- ATM can't see or understand message present on the server
- A function can't see what the called function prints
- They share information using
 - parameter
 - return



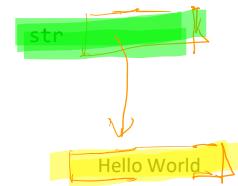
- printed on the server
- we expect to see the message on the ATM
- When it succeeds we just don't want the message "please collect your cash"
 - we want the cash

String class

Tuesday, February 7, 2023 11:18 AM

- String is not a primitive type. It is a class
 - It has reference types.
- Although String is not a primitive type, it is a core language feature and Java gives some additional feature to String that we can't define for our class

```
String str;  
str="Hello World";
```

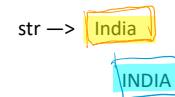


- Java allocates memory just sufficient to store the current string.
- Reference refers to it.

String is java is immutable

- A String object can't be modified after it has been created.
- Any modification to a String requires
 - Creation of a new String object
 - My old reference can now refer to new String object

```
String str="India";  
str.toUpperCase(); //creates a new String at a new memory location
```

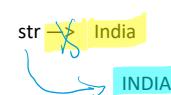


- Note
 - str is still referring to "India"
 - The newly created string is not referenced anywhere and will eventually be garbage collected.

Modifying existing String (reference)

- while we can't modify the string object we can modify the reference to refer to new String

```
String str="India";  
str=str.toUpperCase();
```



- Note
 - str now refers to modified String "INDIA"
 - the original String "India" is now not referred and shall be eventually garbage collected

Why immutability is important!

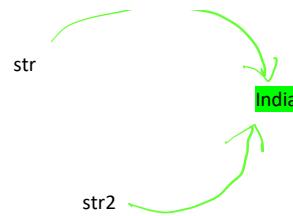
- Since Strings are not modifiable after creation, java can reuse a String object internally

```
String str="India";
```



```
String str="India";
```

```
String str2="India";
```



- compiler realizes that we have same String so instead of storing it twice both reference can refer to the same

- we are sure neither can change it.

== vs equals

- Java String contains a method to compare String content
 - `equals()`
 - return true/false
- This is different from == operator
- == operator compares the references
 - if reference is same the value will naturally be same
- `.equals` compares actual value
 - compare actual characters
 - It is possible that I have two different Strings with same content at two different places.
 - == will return false
 - `.equals` will return true
- We prefer `.equals` over == for String comparison
 -
- equals indicate that two objects have same value
 - tests for value equals (equivalent)
 -
- == indicate that the two references refer to same object
 - indicates they are exactly same.

compares

- `compareTo` compares two String to find how different they are from each other
- It returns the difference for first mismatch character to indicate which comes first in dictionary (unicode sequence)

```
String str1="India";
```

```
String str2="Indonasia";
```

```
int diff = str1.compareTo(str2); // unicode difference of 'i' - 'o' = -6
```

- result interpretation
 - 0 → `equals`
 - negative → first string comes first in dictionary/alphabetic sequence
 - positive → first string comes later in the sequence

Case Sensitive tests

- Normally Strings are case sensitive
 - `"India".equals("INDIA")` → false
 - `"India".compareTo("INDIA")` → 'n' - 'N' = 32
- We have case insensitive comparisons
 - `"India".equalsIgnoreCase("INDIA")` → true
 - `"India".compareToIgnoreCase("INDIA")` → 0

Common String class methods

Method Name	Purpose	Example
-------------	---------	---------

length()	returns length of string	
equals	true/false	
equalsIgnoreCase		
compareTo		
compareToIgnoreCase		
toUpperCase	convert to upper case	
toLowerCase		
charAt	returns character at a given position	
indexOf	searches one string inside another and returns the index of match, -1 if not found	"Indian".indexOf("ia") → 3 "Indian".indexOf("is") → -1
replace	replace a given substring with another in another string. It replaces first match	var s="India is my country. I love India"; s.replace("India","Bharat");
substring	extracts a string from another String using index and length * return string starting from a given index and of specified length	var s="India is my country. I love India"; var x= s.substring(12,7) -->"country" • return string starting 12th index and including 7 characters
format	• static method • creates a new String based on the formatter and arguments supplied	

String Building

Tuesday, February 7, 2023 11:45 AM

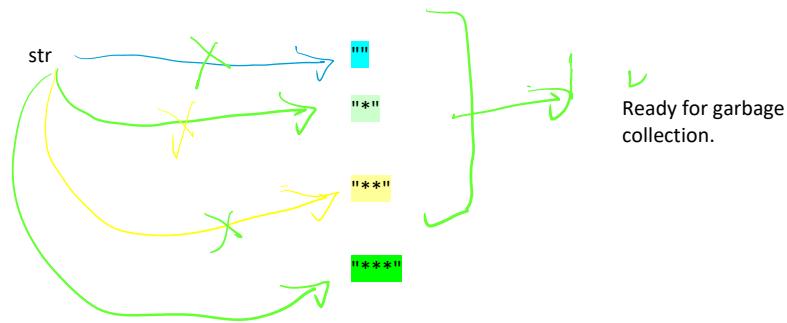
How String concat works?

```
String str= "";
```

```
str+="*";
```

```
str+="*";
```

```
str+="*";
```



Problem

- If we have a large String manipulation, using standard Java String can have extreme performance issue

```
String str="";  
  
for(int i=0;i<10000;i++)  
    str+="*";
```

- If you need large amount of String manipulation consider to use the class `StringBuilder`

StringBuilder

- `StringBuilder` is a class that has methods to manipulate a memory chunk inplace
- It increases memory as per requirement in optimized manner
- Once you have completed the manipulation you can get the final string by calling `toString()` on the builder object.

Array

Tuesday, February 7, 2023 12:04 PM

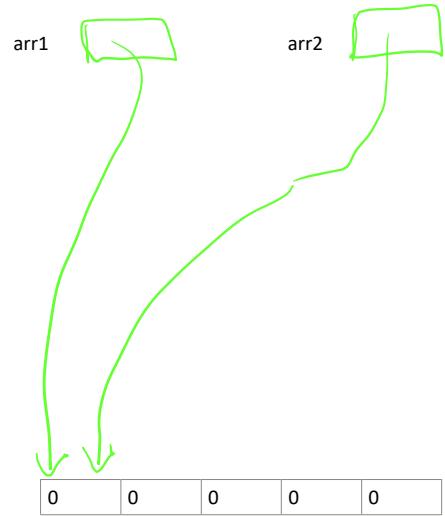
- Array is an Object (reference type) that can hold multiple values accessible by index
 - An array of "int" is still an Object and not a primitive type

Step #1 create an array of int

```
option#1  
int [] arr1 ;  
  
option#1  
int arr2 [];
```

Note

- both the above options are identical
- They create a reference that will refer to an actual array later.
- The size is not specified
 - size belongs to array object and not array reference



Step #2 create the array object

- here we need to specify the size of the array
 - the maximum value it can store

```
arr1 = new int [5];
```

- We created an array of int
- It can store 5 int
- currently the values are all 0.

```
arr2=arr1;
```

- now arr2 also refers to the array object
- No values duplicated
- No second array created.

Creating Array reference and Object together

Option#1

```
int [] list1 = new int [5] ; //array of 5 items all are 0  
  
int list2 [] = {2,3,5,9,2}; //array of 5 items with specified values
```

0	0	3	0	0
---	---	---	---	---

2	3	5	9	2
---	---	---	---	---

Array Access using index

```
list1[2]=3;  
  
System.out.println( list2[2] ); //5
```

Accessing Array Elements using standard for loop

```
for( var i=0; i< list1.length; i++){
    System.out.println(list1[i]);
    list1[1]*=10;
}
```

- Note:
 - int String length() is a method
 - in array it is like a field

Accessing Array Elements (Readonly) using Foreach loop

```
for( var value : list2){
    System.out.printf("%d ",value);
}
//expected output
```

2 3 9 2 6

Note:

- value is the value of list
- it is given one by one
- We don't get index
- we can't modify array element by assigning anything to the value

Problem with traditional Test

Tuesday, February 7, 2023 1:34 PM

1. It is based on print output

- print output is for human eye consumption
- you get an output on the screen
 - there is no way to confirm or deny if what you see is what you expected.
- we must manually maintain a track of what I expect

2. Multiple Test will have multiple output on the same screen

- It is difficult to draw a boundary as to which test printed what message on the output
 - you may get a bunch of true/false
 - which message is associated with which test?

3. Tests may interfere with each other

- often test will be working against an object and may result in
 - false positive
 - test passes despite having an actual bug
 - ◆ bug was not detected
 - false negative
 - test fails despite having no bug

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    String password="p@ss";  
    var amount=20000;  
    var a1=new BankAccount(1, "Vivek", password, amount, 12);  
  
    depositTests("Deposit fails for negative amount", a1, -1, false);  
    depositTests("Deposit succeeds for positive non zero amount",a1,100,true);  
  
    withdrawTests("Withdraw should fail for negative amount",a1,-1,password, BankingStatus.invalidAmount);  
    withdrawTests("Withdraw should fail for wrong password", a1, 1, "wrong password",BankingStatus.invalidCredentials);  
    withdrawTests("Withdraw should fail for insufficient balance",a1, amount+1, password, BankingStatus.insufficientBalance);  
    withdrawTests("Withdraw should pass for happy case",a1,1,password,BankingStatus.success);  
}
```

1. This test is assuming that total balance in the account is amount

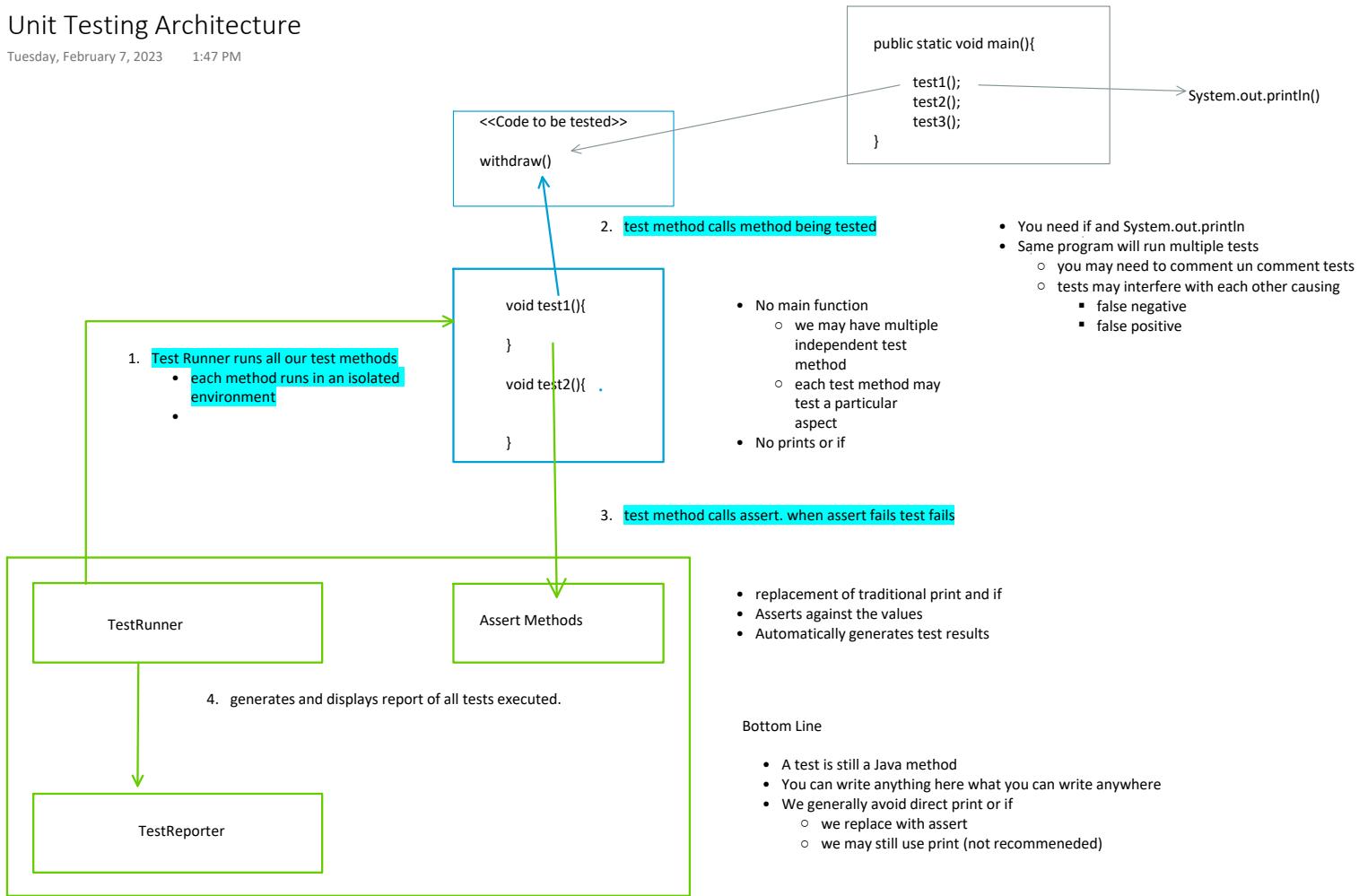
2. However an unrelated test has added Rs 100 to the test object, just violating my assumption and polluting the test data with unexpected value

- The internal logic of withdraw is correct but it is reported as flawed because of an unrelated test condition

FAILED: Withdraw should fail for insufficient balance
expected: insufficientBalance found: success

Unit Testing Architecture

Tuesday, February 7, 2023 1:47 PM

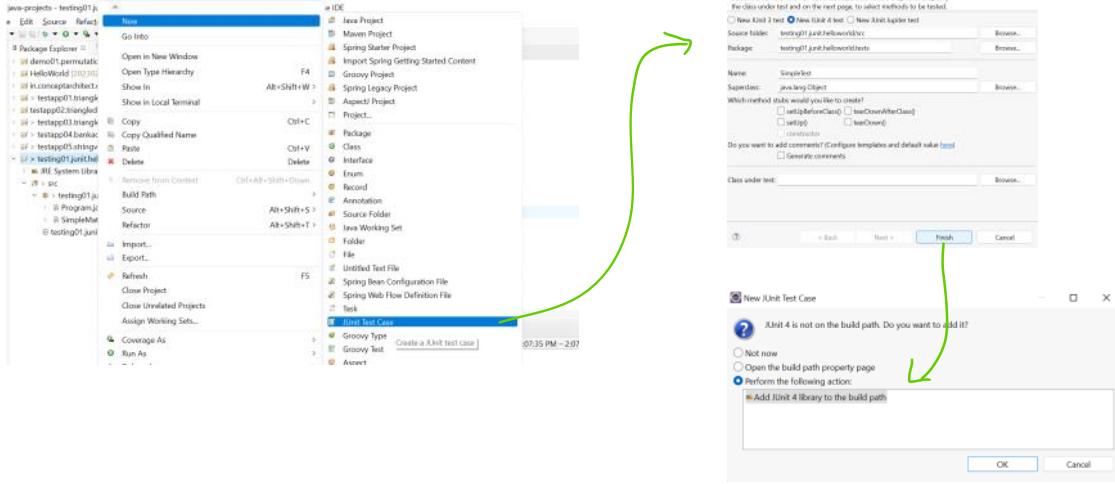


- Forms a typical Test Environment
- There can additional helper library to make tests better and more efficient
- Any of these components can be replaced by third party library
- TestReporter
 - can be as simple as a console output
 - or can be presented in a special window inside your IDE.

jUnit

Tuesday, February 7, 2023 2:01 PM

- jUnit is a third party library created by jUnit for java Unit testing
- This is the first testing library
- It is one of the most popular library under java
 - It is a defacto standard
- We need a separate jar download to make jUnit testing work
- Good News
 - Eclipse comes with full support jUnit including the necessary jar to run it.



Identifying a Test

- A test method is any method that is marked with a @Test annotation
- Eclipse recognizes a class with one or more test method and runs it as a test

```
import org.junit.Test;

public class SimpleMathTests {
    @Test
    public void test1() {
        System.out.println("test1");
    }

    @Test
    public void test2() {
        System.out.println("test2");
    }

    @Test
    public void test3() {
        System.out.println("test3");
    }
}
```

- Any method marked with @Test is a test

What is an annotation?

- Annotation is a Java language feature
- It is a special syntax to introduce meta information in a class
- more on annotation later...

Running first set of Tests

```
import org.junit.Test;

public class SimpleMathTests {
    @Test
    public void test1() {
        System.out.println("test1");
    }

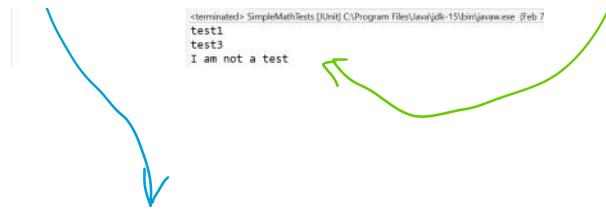
    @Test
    public void test2() {
        System.out.println("test2");
    }

    @Test
    public void test3() {
        System.out.println("test3");
    }

    @Test
    public void notATest() {
        System.out.println("I am not a test");
    }
}
```

Note

1. every method marked as @Test is executed
 - a. name doesn't matter
2. each of these methods run independently and in no particular order
 - a. we may see their output in console



Why did all three test pass?

- They passed because they had no reason to fail.

Assert

- jUnit provides a set of Assert methods present in Assert class
- These method help us "assert" our expectation
 - If our expectations prove correct, test passes
 - else fails

```
public class SimpleMathTests {
    int x=50;
    int y=15;

    @Test
    public void test1() {
        var result= SimpleMath.plus(x, y);
        assertEquals(x+y, result);
    }

    @Test
    public void test2() {
        var result= SimpleMath.minus(x, y);
        assertEquals(x-y, result);
    }

    @Test
    public void test3() {
        var result= SimpleMath.multiply(x, y);
        assertEquals(x*y, result);
    }
}
```

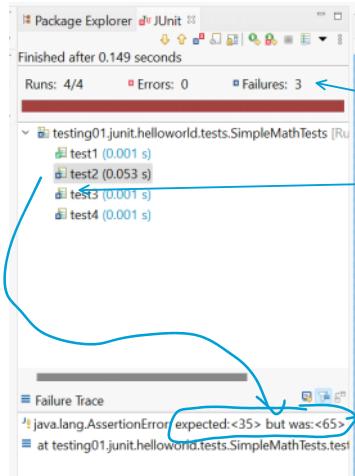
Aside Static Import

- allows us to import a static method from a class

```
import static org.junit.Assert.assertEquals;
```

- Once imported the static method can be used like a global method without needing class reference

```
@Test
public void test1() {
    var result= SimpleMath.plus(x, y);
    assertEquals(x+y, result);
}
```

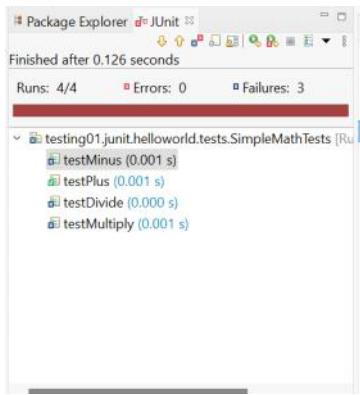


Note

- Here we have three failing tests marked with blue cross
- even if 1 of 100 tests fails you get a brown bar instead of green
- You also get a more detailed report for failure

- report includes the method names for each test
 - what does test2 fail mean?

Use Meaningful Test names



More of Naming

```
public class BankAccountTests {  
  
    String password="p@ss";  
    double amount=20000;  
    double interestRate=12;  
  
    * public void testDeposit() {}  
    * public void testDeposit2() {}  
}  
  
• Generally there will be multiple tests verifying different conditional paths of the same method  
• Example  
    ○ deposit  
        ■ for invalid amount  
        ■ for valid amount  
    ○ withdraw  
        ■ for invalid amount  
        ■ excess amount  
        ■ wrong password  
        ■ happy path  
  
• naming methods with suffix 1,2 may not be clear enough  
    ○ testDeposit1  
    ○ testDeposit2
```

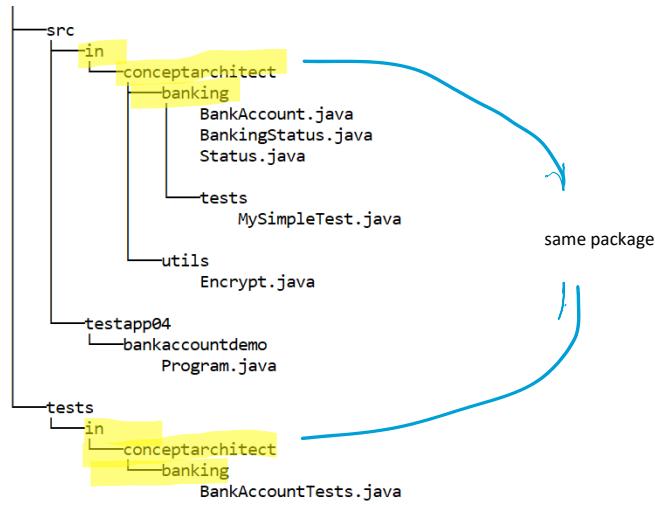
DAMP principle (Descriptive and Meaningful Phrases)

- Method name shouldn't just be a described word, it should be like a phrase
 - It appears in the test report

Unit Testing for package scope

Wednesday, February 8, 2023 9:31 AM

- A package scope members are part of same API
- They can be accessed by other member of the same package but not outsiders
- To test the packages members we can put the test files in the same package
 - The problem is same folder will have
 - application code
 - test code
- To better organize
 - create two sub folders
 - src
 - package in.conceptarchitect.banking
 - ◆ Banking related application classes
 - tests
 - package in.conceptarchitect.banking
 - ◆ Banking related test classes
 - Now add both src and tests in the classpath
 - This way we have to physical folders but one package
 - tests can still access package members of src



BankAccount memory model

Tuesday, February 7, 2023 3:13 PM

```
BankAccount b1=new BankAccount(1,"Vivek", 20000,"p@ss",12);
```

b1

1	Vivek	20000	p@ss	12
---	-------	-------	------	----

```
BankAccount b2=new BankAccount(1,"Sanjay", 20000,"p@ss",13);
```

b2

1	Sanjay	20000	p@ss	13
---	--------	-------	------	----

Problem

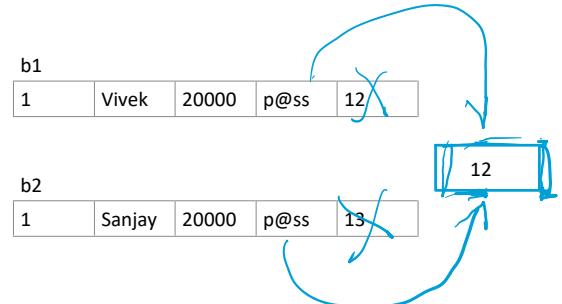
In General all accounts of same type gets same interest

- If we get same interest rate for all object why maintain redundant data?
- How can we have a single shared source of interest rate?
 - global?
- Java doesn't have global variable
- closest candidate is "static" class fields

Static Fields

- Static fields belong to class and not object
- There is a single copy maintained in the memory
- Every object can access this memory
 - no one owns it.
- change of value will impact everyone

```
class BankAccount{  
    int accountNumber;  
    String name;  
    String password;  
    double amount  
  
    static double interestRate;  
}  
  
BankAccount b1=new BankAccount(1,"Vivek", 20000,"p@ss",12);  
  
BankAccount b2=new BankAccount(1,"Sanjay", 20000,"p@ss",13);
```



Assignment 4.1

Tuesday, February 7, 2023 3:31 PM

1. How do I make sure account number is unique?

- write the code to make the account number unique
- write unit test to validate that the account numbers are unique

Composite Output

Wednesday, February 8, 2023 8:18 AM

```
1 public Outcome getBalance(String pass) {
2     var o = new Outcome();
3
4     if (!checkPassword(pass)) {
5         o.setDescription(BankingStatus.invalidCredentials.toString());
6         o.setResult(false);
7     } else {
8         o.setResult(true);
9         o.setDoubleValue(this.balance);
10    }
11    return o;
12}
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1064
1065
1066
1066
1067
1068
1068
1069
1070
1070
1071
1072
1072
1073
1074
1074
1075
1076
1076
1077
1078
1078
1079
1080
1080
1081
1082
1082
1083
1084
1084
1085
1086
1086
1087
1088
1088
1089
1089
1090
1090
1091
1092
1092
1093
1094
1094
1095
1096
1096
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733

```

Static

Wednesday, February 8, 2023 9:35 AM

Static Fields

- A single shared copy is maintained in the memory
- No object owns it
- Everyone can use it.
- Example

```
public class BankAccount {  
    int accountNumber;  
    String name;  
    String password;  
    double balance;  
  
    static double interestRate;  
  
    static int lastId=0;
```

- one copy per object
- belongs to object of the class

- belongs to class and not object
- single copy created for the class
- shared/accessed by every one.

Static Methods

- What is the role of a static method?
- both static and non static methods have single copy in memory
- Difference between static and non-static method

Feature	Non Static Method	Static Method
How to call	• using an object reference	• using class reference • using object reference
this	• contains special this reference that refers to invoking object	• doesn't have this reference as there may not be an object
accessing static members	• YES	• YES
accessing non-static members	• YES	• NO

Why do we need static method?

- We don't need an object to call it
 - Are we sure we are talking about Object Oriented Programming?

```
@Test  
public void interestRateShouldBeCommonToEveryAccount() {  
  
    var a2=new BankAccount(1, "Vivek", password, amount, interestRate);  
  
    //when we change it for a1  
    var newRate= interestRate* 1.05;  
  
    a2.setInterestRate(newRate);  
  
    assertEquals(newRate,account.getInterestRate(),0.001);  
    assertEquals(newRate,a2.getInterestRate(),0.001);  
  
}
```

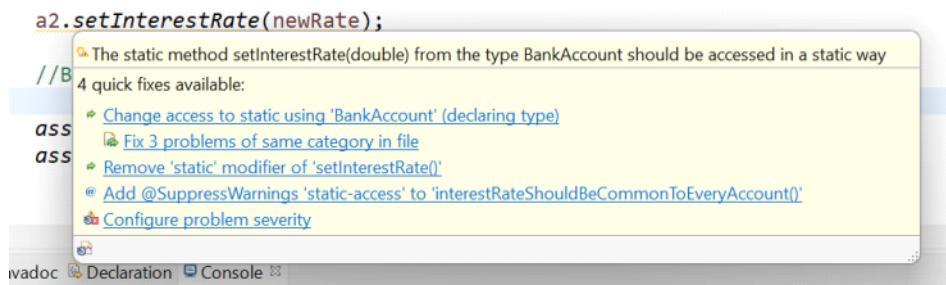
- this code appears to be changing the interest rate for a particular object "a2"
- Actually it is changing for everyone which is not clear by looking at the code
- static will allow you to call this method using class reference

```
//a2.setInterestRate(newRate);

BankAccount.setInterestRate(newRate);
```

Should a class level method be allowed to access by Object reference?

- Java (and c++) allows us to access static methods even using object reference
- But it defeats the purpose of static method
 - Ideally static methods should be used only with class
 - c# doesn't allow accessing static from object reference
- Java best practice guidelines strongly recommends that static methods should be accessed only using class reference and NOT using object reference.



Static Initialization

- Where should we exactly initialize static data members

Option #1 Direct initialization with a constant

```
class BankAccount{

    static double interestRate=12;
    static int lastId=0;

}
```

Problem

- Here the static must be initialized with a constant
- What if I want to read interestRate and lastId from some database
- Now database operation can't be performed this way
- We can't do it in constructor as static fields are not expected to be initialized again and again

Static Block

- Every class can have a static block of code
- This is like a static constructor
- The block is called exactly once, when the class is loaded in memory

```
class BankAccount{

    static int lastId;
    static double interestRate;

    static{
```

```
lastId = BankDb.getLastId(); //may be read from a database  
interestRate=BankDb.getInterestRate();  
//you can write any logic here that you can write inside a method  
}  
}
```

When is static block called?

- It is called exactly once, when the class is loaded in memory
- A class is loaded in one of these three occasions

1. When you create the first object of a class

```
BankAccount account; //class is not yet loaded. creating reference doesn't load class  
account=new BankAccount(); //class is loaded here  
BankAccount a2=new BankAccount(); //class is already loaded. No reload
```

2. When any of the static members are accessed (even if no object is created)

```
double rate = BankAccount.getInterestRate(); //class is loaded. static block executed
```

3. If class is loaded in memory for reflection

```
var cls = Class.forName("in.conceptarchitect.banking.BankAccount"); //static block is executed
```

Do I really need static?

Wednesday, February 8, 2023 10:03 AM

- static means class level and not part of object
 - what is class?
 - description for object (blueprint of an object)
 - if something doesn't belong to object how can it belong to class?
- static means no need of object.
 - Is it Object Oriented.
- interestRate and lastId isn't owned by account object
 - who owns them?

```
public class BankAccount {  
  
    int accountNumber;  
    String name;  
    String password;  
    double balance;  
  
    static double interestRate;  
  
    static int lastId=0;  
  
}  
  
var icici = new Bank("ICICI",12);  
  
var a1= icici.openAccount("Vivek", "p@ss", 20000);  
var a2= icici.openAccount("Sanjay","p2", 40000);  
  
icici.transferFunds(a1, "p@ss", 1000, a2);
```

```
class Bank{  
  
    double interestRate;  
    int lastId;  
  
    int openAccount( String name, String password, double amount){  
        var a = new BankAccount(++lastId, name, password, amount);  
  
        return a.getAccountNumber();  
    }  
  
    boolean deposit( int accountNumber, double amount){  
        return getAccount(accountNumber).deposit(amount);  
    }  
}
```

Test Elements

Wednesday, February 8, 2023 11:02 AM

AAA → Arrange Act Assert

Arrange (Setup)

- setup the initial test condition
- example
 - create the object that we need to test
- It can be done
 - in the beginning of test case
 - @Before
 - @BeforeClass

Act

- execute the function that you want to test
- this is the main activity that we are testing

Assert

- verify code worked as required.

TDD/TFD

Wednesday, February 8, 2023 10:45 AM

- Test First Development or Test Driven development is a paradigm that uses Tests as design specification
 - to underline this idea tdd classes should have a Specs suffix rather than Tests suffix
 - BankTests.java BankSpecs.java
- The idea is we first create a spec file that defines my system's requirement as Test cases
 - At this stage the actual classes is not created
 - Remember it is test first
- Then we create the classes and ensure that it works as per the requirement

TDD Lifecycle → Red-Green-Refactor

Red Phase

- We start with failing test (Red)
- Remember a Test will essentially fail initially as we don't have the classes that can make it pass.
- If we don't have a failing test then it is NOT TDD

- This is the phases where we define design specification that we need to follow.
- Since it is not implemented yet it will fail
- The goal is to provider developer with the system requirement in form of specs files (.java)

Green Phase

- Write the minimal code to make the test pass
- The goal is make the test pass and not write the perfect code
 - You can cheat!!!
- This minimal code may not even be correct or working

- This is not solving the problem but acknowledging the problem
- Here you will get the correct signature of the method that we need to create
- Logic may evolve over a period of time.

Refactor

- Modify the code ensuring that our specs don't break
- With each refactor it may push us to red phase
- We again need to write minimal code to make all tests pass.

Account Types

Wednesday, February 8, 2023 2:59 PM

- A Bank may have different types of Account

Account Type	Min Balance	Max Transactions	Interest Rate	
SavingsAccount	5000	50	standard	
CurrentAccount	0	no limit	0	
Overdraft Account	balance+ OdLimit	50	rate slab 1% extra interest if balance >100000	

- OdLimit is 10% of max historical balance
 - If your historical max balance was 100000 your odLimit is 10% of 100000 = 10000
- If your current balance is 20000 you can withdraw upto
 - $20000 + 10000 = 30000$
- If you withdraw 25000 your balance becomes
 - $20000 - 25000 = -5000$
- There will be a 1% charge on Od
 - 1% of 5000 = 50
- Final balance
 - $-5000 - 50 = -5050$

```
class BankAccount{  
    AccountType type;  
}
```

- you will have to add all logic and information for all account types in a single class
- if we need a new type tomorrow this class will change again.

Inheritance

Wednesday, February 8, 2023 3:07 PM

- Inheritance allows you to extend a class definition by creating a sub class
- The sub class that extends the super class should have a relationship which can be expressed in terms of
 - Sub class is a type of super class
- Example
 - Crow is a type of Bird
 - Car is a type of Vehicle
- We shouldn't extend a class for any other reason/relations like
 - Has A
 - Computer Has a Harddisk
 - Bank has BankAccount(s)
 - Department has Employee
 - Is Like A
 - Crow is Like a Parrot
 - Associated/Works together
 - Computer and Printer works together

Assume we have a class

```
class X {  
    int a;  
  
    public void doTaskA(){  
        ...  
    }  
}
```

Now we can extend this class by sub class Y

```
class Y extends X {  
  
}  
  
• Now Y has all the properties and behaviors of X which it can use  
  
Y y = new Y();  
y.a=20; //works  
y.doTaskA(); //works
```

A sub class object can be referred by a super class reference

```
X v = new Y(); //remember new Y() is a type of X
```

But we don't want a class that is just as good as super class

- Unless you want some additional or changes in the way X object works you don't need to create a sub class

A sub class can add additional property or behavior

```
class X {  
    int a;  
  
    public void doTaskA(){  
        ...  
    }  
}  
  
class Y extends X{  
    int b;  
  
    public void doTaskB(){  
    }  
}  
  
//super class  
class Animal{  
    void eat();  
}  
  
//sub class  
class Tiger extends Animal{  
    void hunt();  
}
```

Animal.super.eat() == new Tiger().

```

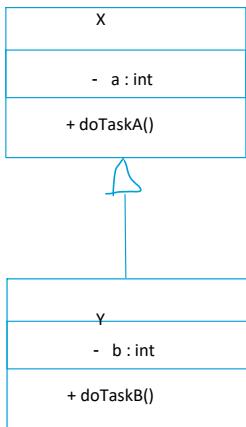
    }
}

}

Animal myFavAnimal = new Tiger()

```

- Now



Inheritance is about creating a relationship: Is A Type Of

- a class Y can extend any class X
- X and Y can be anything
 - Airplane extends Cow
 - Cycle extends Circle
 - RubberDuck extends Rea Duck
 - Computer extends HardDisk
- Inheritance is not about everything
- Don't inherit
 - If there is no relationship
 - don't inherit for relationships like
 - Has A / Owner / Owned
 - Is Like A / Is Similar To
 - Contains
 - Know
 - Associated with
 -



Modifying existing behavior

- Sometimes a behavior defined by the generic super class is not same as what we need in a sub class
 - Example:
 - Mammals generally walk on land
 - Exceptions
 - ◆ Bat is a flying mammal
 - ◆ Whale swims

How do we modify the behavior specific to the sub classes?

- We can modify the behavior we rewriting the same method (with same signature) in the derived class.
- This is known as overriding.

```

class Mammal{
    public String breed(){
        return "Child Bearing";
    }

    public String move(){
        return "Walk on legs";
    }
}

class Bat extends Mammal{

```

```
//overrides the super class move method
public String move(){
    return "Fly";
}

}

class Whale extends Mammal {

    @Override
    public String move(){
        return "Swim in ocean";
    }
}

class Tiger extends Mammal{

    public String hunt(){
        return "Hunt's it's pray";
    }
}
```

IMPORTANT!!!

- Starting Java 6, we have an annotation to mark a method that overrides another method from the super class.
 - This annotation is optional (as it was introduced late) but recommended.
 - Advantage
 - If you write an annotation of a method that doesn't have a correspondence in the super class it will give you a compile time error

```
class Bat extends Mammal{
```

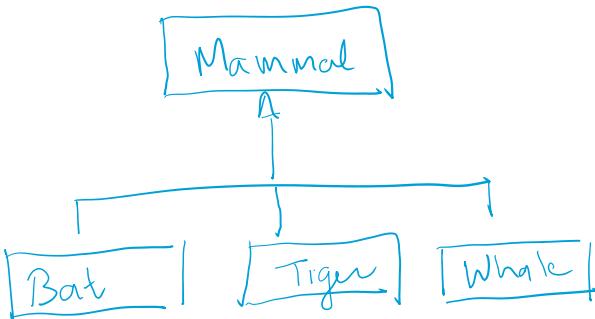
```
//No error. It will be considered as an additional method  
public String move(int speed){  
    return "Fly";  
}
```

}

```
class Whale extends Mammal {
```

```
//considered as an error as no correspoding method present in super class  
//if you want this method to be an additional one, remove the annotation  
@Override  
public String move(int speed){  
    return "Swim in ocean";  
}
```

}



Working with a class Hierarchy

- Since Tiger, Bat and Whale are types of Mammal
 - A Mammal reference can refer to them
 - We can store objects of these types in an array of Mammal

```
Mammal [] mammals = {  
    new Tiger(),  
    new Bat(),  
    new Whale()  
}
```

- Now we can use all these objects in a single for loop

```
for ( var mammal : mammals){  
    System.out.println( mammal.breed());  
    System.out.println(mammal.move());  
}
```

- It will always display "child bearing"
 - It is invoking the breed method defined by the Mammal class and not modified by any subclass

Which method would be invoked here?

- We have two possible candidates
 1. Mammal class method as the reference used is of Mammal
 - we don't know which exact type of object will be used
 - In our case everytime it would be different
 2. The method from the class whose object is being used
 - This can't be determined during compile time
 - It must be decided at runtime only
 - In older compiler driven languages like C++ the default idea was to use method belonging to reference type unless we activate a special mechanism.
 - Java inspects the actual objects at runtime and then decides which method should be called.
 - Java calls this process as **Dynamic Method Dispatch**
 - The more common object oriented term for this behaviour is
 - Polymorphism

Polymerase Chain Reaction

```

Mammal mammal = getRandomMammal();

//Here we (compiler) doesn't know which exact mammal will be returned

var x= mammal.breed() ; //no confusion here. No class Overrides it

var y = mammal.move(); //must select the right move() based on object

• In dynamic method dispatch,
  ○ runtime checks for the object, if it defines/overrides the move method
    □ if yes, that method is called
    □ if no,
      □ it checks for the same in the super class and their super class

• We call the behavior as polymorphism because same move() call may be mapped to different
  objects depending on the dynamic context ( object being used)

```

Inheritance special operator : instanceof

- this is a boolean operator that checks if a given object is an instanceof a given type
- LHS is the object
- RHS is the class

```

tiger instanceof Tiger // true
tiger instanceof Mammal // true
tiger instanceof Animal // true
tiger instanceof Bat // false
bat instanceof Tiger //false

```

Summary so far...

- Any class can extend any other class
- We must extends to achieve an "is a type of" relationship
- Everything (almost) defined in the super class becomes part of the sub-class
- we can add additional attributes (data members) and behaviors (method) in sub class
- we can override an existing behavior of the super class
 - prefer using @Override
- A super class reference (generic reference) can refer to objects of sub class (specific instances)
- When a super class reference refers to a sub class object
 - It can access only those elements that are defined in the super class
 - It can't access any new name or behavior
 - Override is considered as modification of existing behavior and not a new behavior
 - super class reference can polymorphically (dynamic method dispatch) access the overridden method from the sub class

Important

- super class reference can refer to overridden sub class method
- super class reference can't refer to additional behaviors defined by the sub class and not know to super class.

How can Tiger among Mammals Hunt?

```

class Mammal{

    public String eat(){ return "eats food"; }
    public String breed() { return "child bearing";}

    public String move(){ return "walks"; }
}

class Horse extends Mammal{

    public String move(int speed){ return "walks with speed "+speed; } //new behavior
}

class Tiger extends Mammal{

    public String hunt(){ return "Hunts"; }

    @Override
    public String eat() { return hunt()+" and eats" ;}
}

```

```

Tiger tiger =new Tiger(); //this reference can access all Tiger methods both owned and inherited.

@Test
public void tigerCanHunt(){
    assertEquals("Hunts", tiger.hunt());
}

@Test
public void tigerHuntsAndEats(){
    assertEquals("Hunts and eats", tiger.eat());
}

@Test
public void horseCanMoveWithSpeed(){
    int speed=40;
    assertEquals("walks with speed "+speed, horse.move(speed));
}

```

But using Mammal Reference would be different

```

Mammal mammal = new Tiger();

assertEquals ("Hunts and eats", mammal.eat()); //note indirectly mammal invoked hunt also

if(System.currentTimeMillis()%5==0)
    mammal=new Horse();

//which mammal do we have here? A tiger or a Horse? Can we allow it to hunt!!!
var x = mammal.hunt(); //mammal doesn't know how to hunt. It is a method specific to Tiger not available in mammals

```

How do we access the hunt method here

```

Mammal mammal = new Tiger();

//here we are 100% sure my mammal is a Tiger. But Java doesn't know so it refuses to execute
mammal.hunt(); // error

```

We can typecast reference to Tiger to use it

```
((Tiger)mammal).hunt(); //works if it is really a Tiger. Runtime error otherwise.
```

```

Mammal [] mammals = { new Tiger(), new Horse(), new Bat() };

for( var mammal : mammals){
    Tiger t = (Tiger) mammal;
    System.out.println(t.hunt());
}

```

- Works fine for the first loop iteration
- crashes while trying to typecast a Horse into a Tiger
 - This is a runtime error

Always typecast when you are sure it wouldn't crash

```

Mammal [] mammals = { new Tiger(), new Horse(), new Bat() };

for( var mammal : mammals){

    if(mammal instanceof Tiger){
        Tiger t = (Tiger) mammal;
        System.out.println(t.hunt());
    }
}

```

Partial Overriding or Partial Modification

```
class Point{
    int x,y;

    public String toString(){
        return String.format("Point %d,%d ",x,y);
    }
}

class Point3d extends Point{
    int z;

    public String toString(){
        //can I reuse the logic from Point toStrin
        return super.toString() + "," + z; //recursive call to subclass toString
    }
}
```

- sometimes we need a slight adjustment in the core logic that we inherited from the super class
- We can't really have a partial override or modification
 - we either override or we don't
- Once we override, the subclass method will hide the superclass implementation

we can access the super class version of method using super reference

```
class Point{
    int x,y;

    public String toString(){
        return String.format("Point %d,%d ",x,y);
    }
}

class Point3d extends Point{
    int z;

    public String toString(){
        //can I reuse the logic from Point toStrin
        return super.toString() + "," + z; 
    }
}
```

What is not inherited?

- In java when we inherit we inherit all the properties and behavior associated with the super class object.
- We, however, don't inherit the constructor of the class
 - constructor is the creator of the object
 - constructor is not part of the object
- constructor is not inherited because
 - we may need different approach to create object of a sub class
 - we may need to have additional initialization
 - both constructor have different names

A word of "private" members

- private members of a super class is inherited into sub class
- However due to "private" scope, they can't be accessed by any member of the sub class
 - Not even overridden methods
- They can only be accessed using the super class methods that were already accessing it
- This leads to widely accepted WRONG notion that private members are not inherited.
 - There is difference between owning the something and accessing something.
- If a method is private, it can't be overridden
 - It is not accessible

Protected

- protected is a scope for inheritance model
- a protected member is like a private member for the rest of the world
- It is however accessible by the sub classes

Constructor Chaining

Monday, February 13, 2023 9:51 AM

- Although a super class constructor is not inherited, a super class constructor is called alongwith (And before) calling the sub class constructor to create the super class portion

```
class Animal {  
    public Animal(){  
        System.out.println("Animal Constructor called");  
  
    }  
}  
  
class Mammal extends Animal {  
    public Mammal(){  
        System.out.println("Mammal Constructor called");  
  
    }  
}  
  
class Horse extends Mammal {  
    public Horse(){  
        System.out.println("Horse Constructor called");  
  
    }  
}  
  
var horse = new Horse(); //calls constructor of super classes
```

Animal Constructor called
Mammal Constructor called
Horse Constructor called

Note

- Whenever we create the object of the sub class super class constructor is always called
- A super class constructor will execute before executing the sub class constructor
- This process can't be modified.
- By default the sub class constructor always attempts to call the zero argument constructor of the super class
- A code will fail to compile
 - If zero argument constructor is not available
 - If the constructor is not accessible (eg. if it is private)
- A sub class however can specify it it wants a different super class constructor to be called.

```
class Point {  
    private int x,y;  
  
    public Point(int x,int y){  
        this.x=x;  
        this.y=y;  
    }  
}  
  
class Point3d{  
  
    int z;  
  
    public Point3d( int x, int y, int z ) {  
  
        //fails as constructor will try to call 0 argument constructor from super class  
        this.x=x; //fails because x is private  
    }  
}
```

Note

- A sub class constructor must take all parameters required to initialize values, both inherited and new
- User will not be explicitly calling super class constructor
 - They may not even know you inherited something
 - We need to take all parameter

- We can specify which super class constructor it should call

```
class Point3d{  
    int z;  
    public Point3d( int x, int y, int z) {  
        super(x,y);  
        this.z=z;  
    }  
}
```

Note

- when we don't write "super" it is assumed as super()
- if super() is used it MUST be the first statement in the constructor
- You can't write

```
public Point3d( int x, int y, int z) {  
    this.z=z;  
    super(x,y); //MUST BE FIRST STATEMENT  
}
```

Polymorphism

Monday, February 13, 2023 3:22 PM

Different Objects, behaving differently, in the same context is xxxx

Class Oriented vs Object Oriented

Tuesday, February 14, 2023 8:24 AM

```
//Approach A                                         //Approach B

Animal tiger = new Animal (AnimalType.Tiger);          Tiger tiger =new Tiger();
Animal eagle =new Animal (AnimalType.Eagle);           Eagle eagle = new Eagle();
Animal snake = new Animal(AnimalType.Snake);           Snake snake =new Snake();
Animal crocodile =new Animal(AnimalType.Crocodile);    ...

Animal animals[]={tiger,eagle,snake};                  Animal animals [] = {tiger,eale,snake};

for(var animal :animals){
    animal.move();
    animal.eat();
}

package in.conceptarchitect.animals;
enum AnimalType{ Tiger,Eagle,Snake };
class Animal{

    AnimalType type;
    int poisonIntensity;
    Wings wings;

    public Animal(AnimalType type){
        this.type=type;
    }

    public void move(){

        switch(type){
            case AnimalType.Tiger:
                System.out.println("moves on land");

            case AnimalType.Crocodile:
                System.out.println("crawls");
            case AnimalType.Eagle:
                System.out.println("flies");
                break;
            case AnimalType.Snake:
                System.out.println("crawls");
                break;
        }
    }
}

Tiger tiger =new Tiger();

tiger.setType(AnimalType.eagle);

//Approach B

package in.conceptarchitect.animals;
class Animal{
    public void move(){
        System.out.println("Moves somehow");
    }
}

class Reptile extends Animal{
}

class Mammal extends Animal{
}

class Bird extends Animal{
}

class Tiger extends Mammal{

    public void move(){
        System.out.println("moves on land");
    }
}

class Eagle extends Bird{

    Wings wings;
    public void move(){
        System.out.println("flies");
    }
}

class Snake extends Reptile{

    int poisonIntensity;
    public void move(){
        System.out.println("crawls");
    }
}

//package com.anz.animals;
import in.conceptarchitect.animals.Animal;

class Dinasaur extends Reptile{
```

```
}

// client code

import in.conceptarchitect.Animals.*;
import com.anz.animals.Dinasaur;

...

Animal [] animals={new Tiger(), new Dinasaur() };

public void printMammals(Animal[] animals){

}

public void printMammals(Animal[] animals){

    for(var animal:animals){
        if(animal instanceof Mammal)
            System.out.println(animal);
    }
}
```

Animal Hierarchy

Tuesday, February 14, 2023 9:37 AM

```
1 public class Animal {  
2  
3     public String eat() {  
4         return this+" eats something";  
5     }  
6  
7     public String move() {  
8         return this+" moves somehow";  
9     }  
10  
11    public String breed() {  
12        return this+" breeds somehow";  
13    }  
14  
15    public String toString() {  
16        return getClass().getSimpleName();  
17    }  
18  
19 }  
20  
21  
22  
23 }
```

- We don't know the exact implementation details here
- But by returning this information we are actually creating an implementation.

Abstract Methods and Abstract Class

- If we do not have sufficient details to implement a behavior (method), we should mark the method as abstract
 - abstract method tells system that we can't provide implementation details for this method yet
 - It is too generic
 - The implementation shall be provided by some sub class.
- A class that contains one or more abstract methods should also be marked abstract.
 - If class contains non-abstract methods also, it doesn't matter.
 - An abstract class means we don't have sufficient specific information available to create instances of this class
 - Actual instances will be created for the sub class
 - This class is meant only to be a super class

```
2  
3 public class Animal {  
4  
5     public abstract String eat();  
6  
7     public String move() {  
8         return this+" moves somehow";  
9     }  
10  
11    public String breed() {  
12        return this+" breeds somehow";  
13    }  
14  
15 }
```

- If even one method is abstract, the class must be abstract
- When class becomes abstract you can't instantiate it

```
25  
26  
27  
28  
.....  
new Animal(),  
new Dog()  
new Animal()  
new Dog()  
new Dog()
```

Why we create an abstract method when we don't know the implementation?

```
public abstract class Animal {  
  
    public abstract String eat();  
  
    public abstract String move();  
  
    public abstract String breed();
```

- In this case we don't know what Animal will exactly eat.
 - But we know animal will eat.
- If we don't have "eat" method in Animal class we can't call "eat" polymorphically for the sub class.

```
Animal animal = new Tiger();  
animal.eat(); //possible only if Animal class has a eat
```

Why do we need abstract class (Animal) if we can't create an object of this class?

- While we can't create Animal Object (`new Animal()`), we can create

- A reference of Animal that can refer to a sub class Object
- An Array of Animal that can hold objects of sub classes

How does Abstract change class or method behavior

- An abstract method must be overridden by the sub class to provide the necessary implementation
- Any class that extends an abstract class must either
 1. override and implement all the abstract method
 2. or declare itself abstract.

```

4
5 public class Mammal extends Animal {
6
7     public Ma
8         super
9     } * Add unimplemented methods
10    public St
11        return
12    }

```

Abstract by design

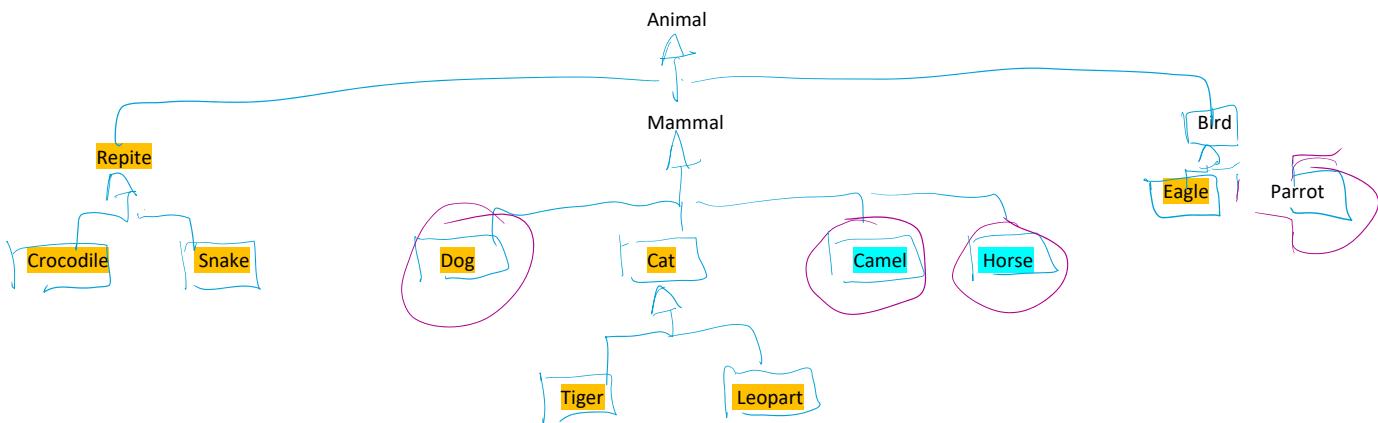
- sometimes a class should be abstract as per design but we don't have any specific abstract behavior
- In such cases we can mark a class abstract even if there is no abstract method in the class

```

3 public abstract class Cat extends Mammal {
4
5     public Cat() {
6         super();
7     }
8
9     @Override
10    public String eat() {
11        // TODO Auto-generated method stub
12        return this+" is a flesh eater";
13    }
14

```

- Now the sub classes will naturally be "concrete or instantiable classe



What is the relationship between animals marked Orange?

- They all have hunt() method
 - They can be called Hunter
 - We may consider that they belong to a super class called Hunter
- A Tiger is an Animal or is it a Hunter?

What is the relationship between objects highlighted blue

- They are Rideable

- They all have hunt() method
 - They can be called Hunter
 - We may consider that they belong to a super class called Hunter
- A Tiger is an Animal or is it a Hunter?
 - Which super class does it belong?
 - Can't it belong to (extend) two super class?
- They are Rideable
- What does purple circle indicate?
- They can be pet animal

Inheritance Restriction

- Java doesn't allow you to extend more than one super class
 - It doesn't support multiple inheritance.
 - You must choose exactly one super class.

Real World Models

- An object can belong to multiple hierarchies
 - Tiger
 - Mammal
 - Hunter
 - Wild
 - Dog
 - Mammal
 - Hunter
 - Pet
 - Crocodile
 - Reptile
 - Hunter
 - Wild
 - Eagle
 - Bird
 - Hunter
 - Horse
 - Mammal
 - Rideable
 - Pet

Interfaces

- An interface is another mechanism to define an object hierarchy
- It is like an abstract class with important changes
 - All methods inside an interface is by default public,abstract
 - You can't use either modifier
 - In latest java releases you are allowed to use public abstract explicitly
 - but you can't use other modifiers like protected
 - There is no concrete or non-public member
 - You implement an interface (and not extend it)
 - just a keyword difference
 - There would be no field or data member in interface
 - If you define it would be final
- While you can extend a single class you may implement any number of interfaces

Remember

- class extends class
- class implements interface(s)
- interface extends interface

interface Hunter{

```
public abstract String hunt();
```

}

- Any class that implements the interface MUST either
 - implement all the interface method
 - declare itself abstract

How can Tiger among Animals Hunt?

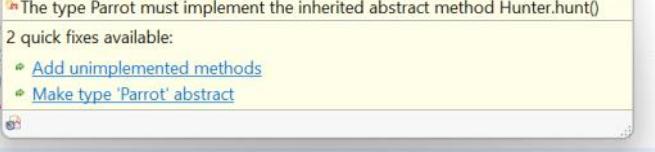
```
if(animal instanceof Tiger) {
    var tiger=(Tiger) animal;
    System.out.println(tiger.hunt());
}
```

How can we check for Hunt for other Animals?

- We essentially don't want to write similar if block for all animals that hunt

Interface Implementation

```
5 public class Parrot extends Bird implements Hunter{  
6     @Override  
7     public String hunt() {  
8         // To...  
9         return "  
10    }  
11 }
```



```
public class Dog extends Mammal implements Hunter{  
    public String eat() {}  
    public String move() {}  
  
    @Override  
    public String hunt() {  
        return this+" hunts in jungle";  
    }  
}
```

How do I model a Pet?

- A Hunter is one who Hunts

```
public interface Hunter {  
    String hunt();  
}
```

- A Rideable is one you Ride on

```
public interface Rideable {  
    public abstract String ride();  
}
```

- A pet is one which ...?

- There isn't any specific behavior that defines a Pet
 - Each pet has a different purpose or behavior

Marker Interface

```
public interface Pet {  
}
```

- A marker interface is an empty interface with no defined behavior
- Implementing classes donot require to override anything
- There is no behavior that can be referred using Pet interface references

What can be the use case of such an interface?

- It is just to define a class hierarchy
- It gives me an ability to test for "instanceof"

Final Method/Class

- A method or a class can be marked final
- If a method is marked final, it can't be overridden by the sub classes

```

1 public abstract class Animal {
2
3     public Animal() {}
4
5     public abstract String eat();
6
7     public abstract String move();
8
9     public abstract String breed();
10
11    public final boolean isPet() {
12        return this instanceof Pet;
13    }

```

```
public class Tiger extends Cat {
```

```
    public boolean isPet() {
```

```
        return true;
```

```
}
```

```
}
```

Cannot override the final method from Animal

1 quick fix available:

Final Class

- if a class is marked final it can't be subclassed

```

4
5 public abstract class Bird extends Animal {
6
7     @Override
8     public final String breed() {
9         // TODO Auto-generated method stub
10        return this+" lays eggs";
11    }
12
13

```

```
interface BankingStatus{
```

```
    String success="success";
    String insufficientBalance="insufficient balance";
```

```
}
```

```
class BankingStatus{
```

```
    public final String success="success";
    public final String
    insufficientBalance="insufficient balance";
```

```
}
```

Object class

Tuesday, February 14, 2023 11:36 AM

- Java has a special class called **Object** class
- This class is the super class of all Java classes
 - predefined
 - userdefined
- Every class directly or indirectly extends Object class
 - A class that extends nothing extends Object
- Example

```
class Triangle extends Object{  
  
}
```

- Class Tiger extends Cat
 - Cat extends Mammal
 - Mammal extends Animal
 - Animal extends Object
- Tiger instanceof Object → true
- any object x instanceof Object → true

What is the advantage of Object class?

1. An Object reference can refer to any object

```
Object o= new Tiger();  
o=new SavingsAccount(...);
```

2. You can put any object into an Object array

```
Object [] universe= { new Triangle(), new SavingsAccount(), new Tiger() };
```

Any method present in Object class is by default available to every Object

- Object class contains a few interesting methods

```
class Object{  
  
    public final Class getClass(){...}
```

```

public int hashCode(){...}

public String toString() { return String.format("%s@%x",
getClass().getName(), hashCode()); }

public boolean equals( Object object) { return hashCode() ==
object.hashCode(); }

public final void wait();
public final void notify();
public final void notifyAll();

}

}

```

Not everything is an Object

- In java primitive types are not classes
- They don't extend Object class
- They don't fall in the hierarchy

```

Object [] values = {
    new Tiger(),      //works
    "Hello World",   //works
    29, //not allowed

};

int x=29;

x instanceof Object

```

Wrapper classes

- to treat "int" as a reference type Object java provides a wrapper class around each primitive type

primitive type	Wrapper Class
int	Integer
boolean	Boolean
char	Character
float	Single
double	Double

- We can use objects of these types to work with Object Hierarchy

```
int x=20;  
  
Integer y= new Integer(x);  
  
Object o1= x; //not allowed  
Object o2 = y; //allowed
```

Auto boxing and unboxing

- java supports implicit conversion between int and Integer

```
int x=20;  
  
Integer y = x; //new Integer(x);  
  
int z = y; // y.intValue()
```

Some standard Java Packages

Tuesday, February 14, 2023 11:52 AM

- **java.lang**
 - The most important of all Java Packages
 - It contains most important classes related to any Java Programming including
 - Object
 - String
 - Math
 - System
 - Primitive Wrappers
 - ...
 - This is so important that all these classes are implicitly imported in all source file
 - It is like we have a pre-written statement

```
import java.lang.*;
```
 - All other standard java packages when used must be imported
- **java.util**
 - A set of assorted utility functions not as important as java.lang
 - we need to explicitly import
 - Important class here is
 - Random
 - Date
 - Collection classes
 - LinedList
 - ArrayList
 - HashSet
 - ...
- **java.io**
 - Classes related to input-output operations that may be
 - file i/o
 - network i/o
 - memory i/o
 - File and Directory management
- **java.net**
 - Related to network programming
 - TcpSocket
 - UdpSocket
- **java.awt**
 - For developing java desktop application

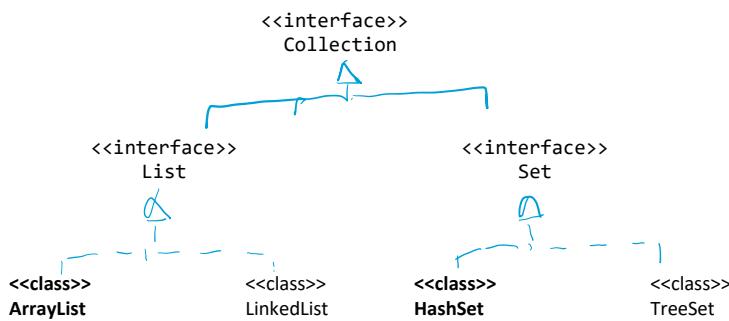
Java Collection Classes

Tuesday, February 14, 2023 12:02 PM

- Java provides a set of dynamic collection classes
- Think of them as advanced version of Array that can expand infinitely
- They have additional features
- Java Collection classes came with original version of Java and got major improvements after Java 6

Pre Java 6

- Java collections classes were collection of Objects
- This way they can hold any object inside



- **ArrayList**
 - It is a dynamic array
 - It is expanded in an optimized way
 - One of the most popular collection
 - Memory is internally continuous
 - good to access random values programmatically
- **LinkedList**
 - Uses double linked list algorithm
 - data is stored in non-contiguous nodes
 - good choice if you need to insert values in between collection
- **TreeSet**
 - stores unique set of values
 - Values are stored in sorted order using Binary Search Tree algorithm
 - It is a better choice if you need sorted set of values
- **HashSet**
 - stores unique set of values using hashing algorithm
 - values are not sorted
 - It is optimized for fast search of data inside the collection

```
interface Collection{
    void add(Object o);
    boolean contains(Object o);
    void remove(Object o);
    void clear();
    int size();
    Object [] toArray();
    Iterator iterator();
}

//linear indexed collection
interface List extends Collection{
    void insert(int index, Object value);
    void set(int index, Object value);
    Object get(int index, Object value);
    void removeAt(int index);
}

interface Set extends Collection{
    //no additional method
    //defines collection of unique values
}
```

A simple example

```
var accounts= new ArrayList();

accounts.add (new SavingsAccount(...));
accounts.add(new CurrentAccount(...));
accounts.add(new OverdraftAccount(...));

...
//accessing using standard for loop
vor( int i=0; i<accounts.size(); i++){
```

```

Object a= accounts.get(i); //we get Object
a.creditInterest(); //not allowed as Object doesn't know creditInterest

if(a instanceof BankAccount){ //skipped as we stored only BankAccount sub type
    BankAccount account= (BankAccount) a;
    account.creditInterest(); //works
}

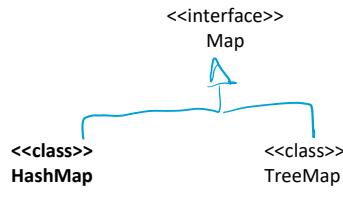
}

//also supports for-each type loop
for(var a : accounts){

    var account= (BankAccount) a;
    account.creditInterest(12);
}

```

A collection that is not a Collection



//Map interface represents a collection of key value pairs
//where both key and values are Objects

```

interface Map{
    void put(Object key, Object value);
    Object get(Object key);
    void remove(Object key);
    boolean contains(Object key);
    void empty();
    Set keys();
    Collection values();
}

```

- this of Map as an array that doesn't have integer index but can have any other type index
- Keys are unique
- putting multiple values against the same key overwrites the previous value without warning

```

//Example: Storing Country Information

HashMap db=new HashMap();

db.put("IN", new CountryInfo("India", "New Delhi", "INR", ...));
db.put("FR", new CountryInfo("France", "Paris", ...));
...
db.contains("IN"); //->true
db.contains("XYZ"); //->false

for(var key : db.keys()){
    System.out.printf("%s : %s\n", key, db.get(key));
}

db.put("IN", new CountryInfo("Bharat", ...)); //replaces "India"

```

IMPORTANT!!!

- Since Collection and Map stores Object they can work with any Object but NOT with primitive types
- To use primitive type you need to use Object wrappers

```
var numbers =new ArrayList();
```

```
numbers.add( 29); //numbers.add (new Integer(29)) --> autoboxed to Integer
numbers.add( 40);

int x= (Integer) numbers.get(0);
```

Generics

Tuesday, February 14, 2023 1:37 PM

- Generics is a programming paradigm to create algorithms that are independent or agnostic of the data type they operate on
 - Example: Collection like ArrayList
 - It is expected to store some value
 - But storage and retrieval doesn't care about "what exactly" you store
 - All we need to know is it is an "Object"

Problems with traditional Object based Generics

- When we create an object, we intend to store specific type of value in this collection
 -

```
ArrayList accounts = new ArrayList();
```

- Here we expect to store BankAccount objects

```
accounts.add(new SavingsAccount(...));  
account.add(new CurrentAccount(...));
```

- Problem #1 compiler not detecting the problem is a problem.

- But we want to store BankAccount is known to us, not to Java compiler or runtime
 - we can store anything we want

```
account.add(new Tiger()); // illogical but syntactically perfectly
```

- Problem #2 when fetching the value it is retrieved as Object and not as Bank Account

```
for( var a : accounts){  
    a.creditInterest(12); // fails. Object reference doesn't have creditInterest  
}
```

- Solution to Problem#2 : explicit typecast

```
for( var a : accounts){  
    var account = (BankAccount) a;  
    account.creditInterest(12); // fails. Object reference doesn't have creditInterest  
}
```

- Problem #3 (caused by Problem #1 and Problem #2)

- We can't restrain a collection object to store a particular type only
 - we can store anything
 - when typecasting it will throw exception

```
ArrayList accounts = new ArrayList();
```

```
accounts.add(new SavingsAccount(...));
```

```

account.add(new CurrentAccount(...));
account.add(new Tiger()); //false positive

for( var a : accounts){
    var account = (BankAccount) a; //false negative
    account.creditInterest(12); //fails. Object reference doesn't have creditInterest
}

```

- code will crash on this line
- because of an error in this line
- Not only code is failing it is reporting a failure at a wrong place
 - You get a false negative here

Problem #3 solution → guarded typecasting

```

for( var a : accounts){
    if( a instanceof BankAccount){

        var account = (BankAccount) a; //false negative
        account.creditInterest(12); //fails. Object reference doesn't have creditInterest
    }
}

```

- This is not a clean solution
 1. we shouldn't be needing this check
 - afterall aren't we suppose to store only accounts
- We may need to write similar if block through out application
 - withdraw
 - deposit
 - transfer
- What should I do for bad/incompatible object here?

Java 6 Generics

- Now we have special syntax called generic syntax in Java
- We define our class or object in terms of abstract data type and not specific one

Step #1 We need an Array of BankAccount

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
```

- Now accounts is an ArrayList of BankAccount type only
- Java compiler is now aware that only BankAccounts can be stored

Note

- this is a Java 6 generic syntax
- ArrayList class is created to store a generic value and NOT specific type
 - Not even Object
- We should (must) inform Java about what Kind of value I want to store in my ArrayList
- This is not an ArrayList or Collection Specific feature
 - we can also create our own Generic designs

Step #2 adding values to collection

```

accounts.add( new SavingsAccount(...)); //works because SavingsAccount is a BankAccount
accounts.add(new CurrentAccount(...));

accounts.add( new Tiger()); //compile time error!!!

```

Step#3 accessing the value back

- since array List knows that we are storing BankAccount it returns BankAccount reference and Not Object reference

```

for( var a in accounts){

    //here var -> BankAccount

    a.creditInterest(); //works fine. Not typecast. No Error
}

```

```
}
```

Generic parameter can be any class/interface but not primitive

```
ArrayList<Hunter> searchAllHunters(ArrayList<Animal> animals){  
    ArrayList<Hunter> hunters=new ArrayList<Hunter>();  
    for(Animal animal : animals){  
        if(animal instanceof Hunter){  
            hunters.add((Hunter)animal);  
        }  
    }  
    return hunters;  
}
```

- we can't have primitive type as parameter

```
ArrayList<int> numbers; //not allowed
```

- But we can use wrapper type

```
ArrayList<Integer> numbers;
```

Generic Maps

- Simple HashMap → storing CountryInfo against Country code

```
HashMap<String,CountryInfo> db=new HashMap<String,CountryInfo>();
```

Handling complex Generic Parameters

- Consider a map where
 - key is Department
 - value is an ArrayList of employees

```
HashMap<Department, ArrayList<Employee>> db = new HashMap<Department, ArrayList<Employee>>();
```

- We can handle it in two different ways
- Java 7 syntax

```
HashMap<Department, ArrayList<Employee>> db = new HashMap<>();
```

- Auto detect from the LHS

- Java 9 syntax

```
var db = new HashMap<Department, ArrayList<Employee>>();
```

- auto detect from RHS

- Don't use both
- Will not work

```
var db=new HashMap<>();
```

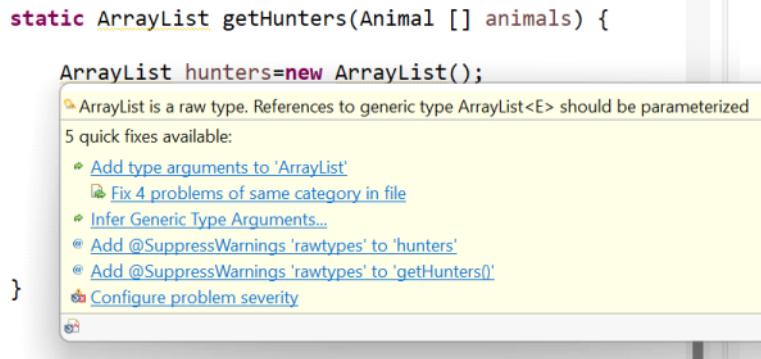
Generic Parameter if not supplied defaults to Object

```
ArrayList list =new ArrayList(); //this code will give warning messae.
```

- is same as

```
ArrayList<Object> list= new ArrayList<>();
```

- This first syntax is not recommended in modern code
- It is just for backward compatibility from pre-generic releases
- If we really want an array List of Objects we should use the second explicit syntax



Generic Part 2

Wednesday, February 15, 2023 12:14 PM

A Tiger is an Object. But a List<Tiger> is not a List<Object>

```
List<Tiger> tigers = new ArrayList<Tiger>();
```

```
List<Object> objects = tigers; //not allowed.
```

Why is it not allowed?

- If it is allowed, then we can write

```
objects.add(new Triangle()); //now a triangle is added to a List<Tiger>
```

- This will invalidate a List<Tiger>

Where can I use generic

- A generic can represent an object or a method where the algorithm doesn't need to know actual data type
- It can think of the data type to be just like Object
 - It knows methods present in Object class
- It can't directly access behaviors defined for a given type

Assignment 7.1

Tuesday, February 14, 2023 2:52 PM

- Write a method to search a List and return a List of all numbers that are divisible by 3

```
class Search{
    List<Integer> searchDivisibleBy3(List<Integer> values){
        var result= new ArrayList<Integer>();
        for(int value : values){
            if(value%3==0){
                result.add(value);
            }
        }
        return result;
    }
}

class Test{
    var list = Arrays.asList( 2, 9, 8, 4, 11, 3, 2, 17,6);
    @Test
    public void searchCanSearchAndReturnNumbersDivisibleBy3(){
        Search s=new Search();
        var result = s.search(list);
        //assert
        for( var value : result)
            assertTrue(value%3==0);
    }
}
```

```
class Search{
    List<Integer> searchDivisibleBy3(List<Integer> values){
        var result= new ArrayList<Integer>();
        for(int value : values){
            if(value%3==0){
                result.add(value);
            }
        }
        return result;
    }

    List<BankAccount> searchCurrentAccountsWithBalanceAbove5000(List<BankAccount> values){
        var result= new ArrayList<BankAccount>();
        for(var value:values){
            if(value instanceof CurrentAccount && value.getBalance()>5000)
                result.add(value);
        }
        return result;
    }

    List<Animal> searchPetMammals(List<Animal> animals){
        var result=new ArrayList<Animal>();
        for(var value :values){
            if(value instanceof Mammal && value.isPet())
                result.add(value);
        }
        return result;
    }
}
```

Important Steps In Search

1. Make a Result list
2. Loop through the original list
 - a. check if the current value is a match
 - b. Add the current value in the result
3. Return the result

Note the methods are performing two Job

1. How to Search a.k.a Core Search Algorithm
 - A Generic Logic
2. What to Search
 - This is specific to situation
 - Keeps changing very frequently
 - Because of this part the search method can't be generic to serve all needs.

Solution —> Apply DRY

1. Encapsulate whatever repeats
2. Abstract Whatever changes

3. Inject abstraction in encapsulation

```
List<Integer> searchDivisibleBy3(List<Integer> values){
    var result= new ArrayList<Integer>();
    for(int value : values){
        if(value%3==0){
            result.add(value);
        }
    }
    return result;
}
```

```
List<BankAccount> searchCurrentAccountsWithBalanceAbove5000(List<BankAccount> values){
    var result= new ArrayList<BankAccount>();
    for(var value:values){
        if(value instanceof CurrentAccount && value.getBalance()>5000)
            result.add(value);
    }
    return result;
}
```

1. Encapsulate whatever repeats

```
List<Object> search_____(List<Object> values){
}
```

- doesn't work because List<Object> reference can't accept List of other types

Java Generic Syntax

- we can write our own generic function which can take a List of any generic type X and return a List of the same generic type X
- X is indicated as a prefix to the generic method

```
<X> List<X> search_____(List<X> values){
    var result= new ArrayList<X>();
    for(X value : values){
        if(value%3==0){
            result.add(value);
        }
    }
    return result;
}
```

Rule#2 Abstract whatever changes

- what is the abstraction for
 - value%3==0
 - value.getBalance()>5000
 - value.getCost()>2000
 - value.getAuthor().equals("Vivek Dutta Mishra")
- what are we doing here?
 - MatchCriteria

```
<X> List<X> search(List<X> values,Matcher<X> matcher){
    var result= new ArrayList<X>();
    for(X value : values){
        if(matcher.match(value)){
            result.add(value);
        }
    }
    return result;
}
```

- Who will do the match (which object)
 - search object?
 - It should be other class
 - It should be abstract

Where will I get this matcher object?

Rule #3

- we can pass the matcher as a parameter

What is a Matcher?

```
interface Matcher<T>{
    boolean match(T value)
}
```

Object Oriented Programming Recap

Wednesday, February 15, 2023 8:13 AM

There are two Key terms

1. Object

- Represents a domain (problem space) entity
- It may be something like a real world object having
 - property/attribute/state
 - behaviors
 - Responsibility
- Idea is to treat the program as a set of interacting objects

2. Class

- Generally class is called blueprint/template of an object
- Class → Classification and Sub Classification
 - The basis of these classification is object's properties and behavior
 - Forms a hierarchy
- Class is conceptually optional
 - It is not optional in Java
 - There are Object Oriented Langauge with no notion of class
 - Java Script

Object Oriented Modeling Elements

Encapsulation

- binding state and behavior together to model a **responsibility**
- we achieve this by
 - ensuring states can't change in an invalid way
 - generally guard them from external access using scope rule or conventional approach
 - Define proper getter/setter/property to access them adhering to the business/domain requirement
- An object **has/owns** states that helps it perform the expected job
- Defines an element of **Reuse**
- **More Responsible => More Usable**

Inheritance

- Defining a hierarchy of class and sub-class
- It defines "Is A Type of Relationship"
- Inheritance Helps Us in
 - **Helps Us Reuse**
 - **Models Parent-Child Relationship**

- Try to avoid using inheritance for "reusability"
 - We should mostly inherit from abstract class / interface
 - where there is little to re-use
- What is the role of inheritance or class hierarchy?
 - define a group of similar entities that can be substituted for each other
 - Allows us to create a design where a component can be replaced with another
 - because requirement changes.
- Inheritance is a mechanism to implement "polymorphism"

Polymorphism

- Allows us to create components that implement a common standard (abstract class/inheritance)
- We use the concrete component as an implementation of same common standard
 - We don't code against concrete class but against the abstract idea
 - We have a holder (socket) where we can add a Bulb
 - Bulb is something that can give light
 - Exact implementation technology doesn't matter
 - ◆ It can change in future as long as the interface (method signature) doesn't change.
- Super class or interface defines abstract (we know what to do but not how) standard
- Sub class defines the exact implementation details (how)
- While we use the sub class object, we focus on the super class core concept where we plug in different implementation.

Parent Child Inheritance (Real World)

Wednesday, February 15, 2023 8:41 AM

Model A

```
class Father{  
    ...  
}  
  
class Son extends Father{  
    ...  
}  
  
Father aman = new Father(...);
```

```
Son ajit =new Son(...);
```

Model B

```
class Person{  
    String name;  
    Person father;  
}  
  
Person aman = new Person(...);  
  
Person ajit =new Person("Ajit",aman);  
  
//how ajit gets property from aman?  
  
bank.transfer(  
    aman.getAccount(),  
    amount, password,  
    ajit.getAccount()  
);  
  
ajit.setDNATFrom(aman);
```

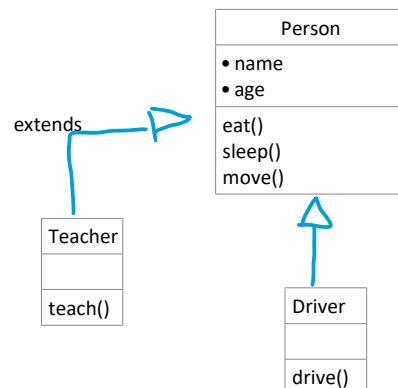
Quiz: A father has 3 childs and 3 lac Rs in property. How much each child will get
(Assume No biase)

Takeaway

Real World Inheritance	OOP Inheritance
Object to Object relationship	Class to Class Relation Ship
Both parent child are objects generally of same type • parent or a Dog is a Dog not Mammal	Doesn't represent Parent-Child Relationship Represents Type and Sub Type relationship • Dog extends Mammal to represent Dog is a sub type of Mammal
Represented by transferring property from one object to another	Information is shared by generic super class to specific sub class to define their behavior

Representing Person

Wednesday, February 15, 2023 8:57 AM



```
var vivek = new Teacher(); //new Person();
```

```
var prabhat=new Person();
```

Is Vivek a teacher or driver?

```
class Vivek extends Person implements
Teacher,Driver{
```

```
}
```

```
var vivek=new Vivek();
```

teach()



drive()

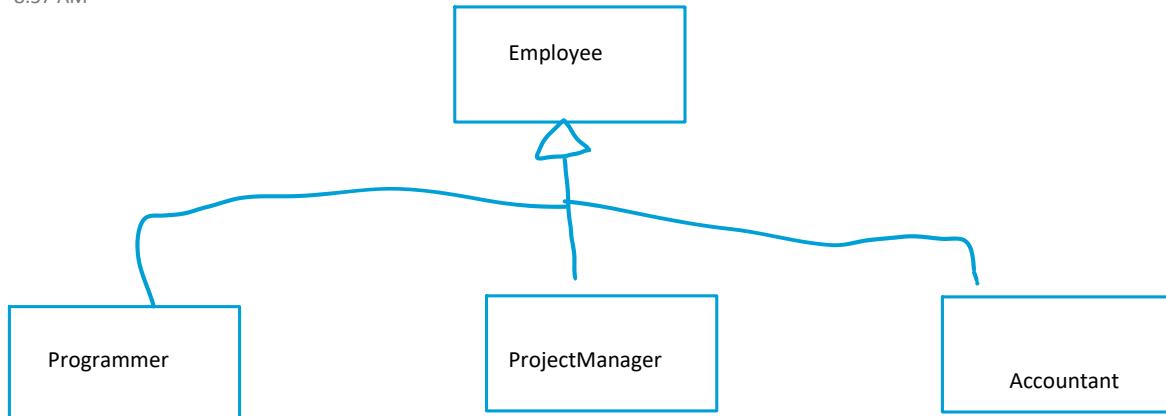


Vivek

Prabhat

Employee Management System

Wednesday, February 15, 2023 8:57 AM



```
Programmer p=new Programmer("Rajiv Bagga");
//how do I promote the programmer to a project manager
ProjectManager pm=p; //incompatible reference
ProjectManager pm=(ProjectManager)p; //fails
```

Is A vs Has A

Wednesday, February 15, 2023 9:15 AM

Prefer Has A over Is A

- Whenever possible try to model a relationship as "Has A" instead of "Is A"
- Why?
 - Has a is more dynamic, runtime, re-usable and scalable relationship
- Can we really convert "is a type of" to "has a"

```
var rajiv = new ???();  
rajiv.setEmployement(...);
```

Use Case #1: Rajiv is an Employee

- Can I Convert
 - Rajiv is an Employee to
 - **Rajiv Has an Employee**
 - It becomes a different idea
 - **Rajiv Has an Employment**
 - Often when changing from is a to has you may have to change the naming.
- **What is the advantage?**
 - if "rajiv" is an employee
 - rajiv is always an employee
 - no retirement
 - no self employment
 - if rajiv has an employment
 - rajiv.setEmployement(null); //just left the job
 - rajiv.setEmployement(new SelfEmployment());

```
class _____{  
    //Employement employment;  
    List<Employement> employments;  
}
```

Inheritance is a no-scalable relationship

- If Rajiv is an employee he is just "one" instance of employee
 - Not object can be multiple instances of a class
- If Kent Clark has an employment
 - Can be journalist
 - Can be Superman

IMPORTANT

- Many "Is A" relationship is badly understood as a relationship.
- Anand is a Doctor or
 - Anand has the role of a Doctor
- Anita is a Mother
 - Anita has a relationship of being mother to ...
- Vivek is a teacher or a driver?
 - Vivek has the role of a teacher or driver

```
vivek.setRole( teacher);  
vivek.work(); //works as teacher
```

Who is Vivek?

- **Vivek is a Human?**
- Vivek has human qualities
- **Vivek is a Person**

```
vivek.setRole(driver);  
vivek.work(); //works as driver
```

```
vivek.setRole(new Author());
```

Object Oriented Design Principles

Wednesday, February 15, 2023 10:49 AM

1. Open Close Principle (OCP)

- This can also be considered as the key goal of any software design
- It suggests that design should be
 - **Open for extension**
 - There will be future changes
 - We should create a future proof design that can accommodate
 - new additions
 - modifications
 - removal of old features
 - **Every program is by nature "open" as long as you have the source code**
 - You may convert a Cow to an Airplane if you wish!!!
 - **Close for modification (source level)**
 - When a change comes, it shouldn't be by changing existing source code
 - Every future change should be adding new class rather than modifying the old one.
 - change should additive.
- **Why is important to Close the code for source level changes?**
 - Every change triggers a cycle of compile-test-deploy-distribute
 - changes are expensive.
 - A change may introduce bugs
 - A change may not be acceptable to every stakeholder
 - changes are not always from version 1 to version 2
 - it may be version A and version B.

OCP

- Open for modification
 - If there is a bug in your code feel free to modify it and correct
- Close for extension
 - Do not add additional features to the code.

Single Responsibility Principle

- A single responsibility
- One reason to exist
- One reason to change
- Guidelines for Achieving SRP
 1. Use Meaningful name
 - without meaningful name responsibility can't be ascertained
 - Fish Swim → valid responsibility
 - Fish fly → invalid responsibility
 - Foo bar → unclear
 - Avoid names joined with and/or
 - Example
 - ◆ class IncomeAndServiceTaxCalculator
 - ◆ void createAndSave(){}
 - ◆ void insertOrUpdate(){}
 - Avoid using abstract names for concrete class
 - Use Abstract name for abstract class / interfaces
 - More concrete names for concrete classes
 - ◆ class TaxCalculator
 - ◊ Is it violating SRP?
 - ◊ I am still doing IncomeTax calculation and Service Tax calculation
 - TaxCalculator is a generic for an concrete class
- 2. You should have fewer behaviors
 - Single Responsibility doesn't mean single behavior
 - But it means a set of co-hesive behavior
 - Example#1 Printer
 - should have
 - ◆ print()
 - ◆ cancel()
 - ◆ ejectPaper()
 - shouldn't have
 - ◆ scan()
 - Example#2 Car
 - should have
 - ◆ start()
 - ◆ move()
 - ◆ turn()

- ◆ stop()
 - shouldn't have
 - ◆ drive()
 - ◊ car's don't drive themselves
3. Cohesive Design
- Most of your methods should access most of the fields most of the time
 - PrintingDevice
 - fields
 - ◆ ink
 - ◆ roller
 - ◆ scanning surface
 - methods
 - ◆ print()
 - ◆ scan()
 - ◆ eject()
 - ◆ cancel()
- avoid mutually exclusive codes in the same class/method
 - avoid parameters that are always null in a given context

3. DRY Principle (Don't Repeat Yourself)

- avoid redundant code in your design
- redundant codes are generally violation of SRP
- when the code core functionality changes we need to change at multiple places
- Solution
 1. Encapsulate whatever repeats
 2. Abstract whatever changes (as abstract class/interface)
 3. Inject abstractions in the encapsulated unit

4. Interface Segregation Principle

- Avoid FAT interface (interface with too many methods)
- Fat interface => Fat Class => Violation of SRP
- An interface should have only as many methods that every implementor would like to implement
 - avoid optional methods
 - avoid mutually exclusive methods
- Solution
 - Let interfaces with optional method extend interface with core method
 - A class may implement many interfaces

5. Liskov's substitution Principle (Definition of Polymorphism)

- A client that can use a super class component can use the sub class component without any problem
- A subclass shouldn't introduce breaking changes
 - If it works for super class, it works for sub class
 - breaking changes may mean a bad class hierarchy
- Advantage
 - When a new changes comes, it can be brought as a sub class
 - open for extension
 - a new class (subclass) introduces the required changes
 - it can override existing methods
 - closed for modification
 - client doesn't need to change
 - current component (super class) doesn't need to change
 - IMPORTANT!
 - In newer languages including java it is difficult to break LSP
 - In Java a method can be overridden in a more relaxed scope but not restrictive one
 - ◆ a protected superclass method can be overridden as public in subclass
 - ◆ but a public super class method can't be overridden as protected or private in sub class
 - ◊ If a method exists in super class it exists in sub class. It can't be removed.
 - BUT LSP can still be broken
 - ◆ to be discussed later.

Dependency Inversion Principle

- What is dependency?

- Knowledge is ownership
 - If you know something you own it.
- Knowledge is dependency
 - When something that you know (depend) changes, it may change you also.
 - Dependency causes a cascading change
- Dependency Inversion Principle says
 - A concrete class X instead of depending (knowing) another class A should depend on a common abstraction
 - client uses abstraction
 - component implements abstraction
 - What you depend on should be abstract
- Advantage
 - we can plug in any concrete implementation
 - we can switch the implementation whenever we wish

S	R	P
O	C	P
L	S	P
I	S	P
D	I	P

|

Car vs SelfDriven Car

Wednesday, February 15, 2023 11:36 AM

How do I define Car and Self Driven Car?

- Normal cars don't have drive()

```
class Car{  
    public void start(){...}  
    public void stop(){...}  
    public void move(){...}  
    public void turn(){...}  
}  
  
class SelfDrivenCar extends Car{  
    public void drive();  
}
```

```
class Car{  
    Driver driver;  
    public void start(){...}  
    public void stop(){...}  
    public void move(){...}  
    public void turn(){...}  
}  
  
interface Driver{  
    void drive();  
}  
  
class HumanDriver implements Driver{  
}  
  
class RoboDriver implements Driver{  
}
```

Nested Inner and Anonymous Classes

Wednesday, February 15, 2023 2:08 PM

- Sometimes we need a class that will be used by only one class
- It may be for the internal implementation logic
- This class may not be required by others.
- In such cases we can write a class inside another class

```
13
14 public class CollectionsTests {
15
16     class GreaterThanMatcher implements Matcher<Integer>{
17
18         int threshold;
19         public GreaterThanMatcher(int threshold) {
20             this.threshold=threshold;
21         }
22         @Override
23         public boolean match(Integer value) {
24
25             return value > threshold;
26         }
27     }
28
29
30
31
32     @Test
33     public void searchCanReturnAllValuesGreaterThan20() {
34
35         var result = cu.search(numbers,new GreaterThanMatcher(20));
36
37         assertEquals(2, result.size());
38     }
39 }
```

Nested vs Inner class

- If a inner class is static it is called Nested class
- If a nested class is static it can be referenced using class reference
- if it is non-static it can be referred only using Objects

Nested/Inner class still creates a separate class file

- the class file will have a name like
 - OuterClass\$InnerClass.class

Name	Date modified	Type	Size
CollectionsTests\$GreaterThanMatcher.class	2/15/2023 2:08 PM	CLASS File	2 KB

```
class Outer{
    public class Nested{
    }

    public static class Inner{
    }

    @Test
    public void canCreateInnerObjectFromOutside(){
        var obj= new Outer.Inner();
    }

    @Test
    public void canCreateNestedObjectFromOutside(){
        var obj=new Outer.Nested(); //Not static and not allowed
        var outer=new Outer();
        var nested = new outer.Nested();
    }
}
```

Anonymous class

- sometimes a class is needed only by a single method
- In such cases we can create a class directly inside the method
- These classes are created as anonymous class
 - They will have no name
 - There will be a single object of this object
 - This object must extend some class or implement some interface

```
public void searchCanReturnAllSavingsAccount() {
    var accounts=createAccounts();

    var matcher= new Matcher<BankAccount>() {
        public boolean match(BankAccount value) {
            return value instanceof SavingsAccount;
        }
    };

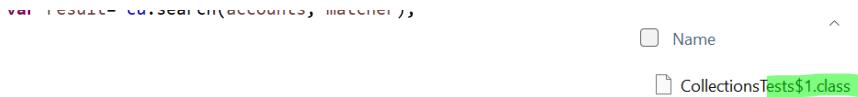
    var result= cu.search(accounts, matcher);
}
```

One "new" creates two new Things!

1. create a new class that implements Matcher<BankAccount>
 - This class is never called again
 - It doesn't need a name
2. create a new Object of this matcher class

Even Anonymous classes are created as separate .class file

Name



Java 8 Lambda Functions

- Lambda functions are like anonymous class Object
- They work for only interfaces that have a single interface method
- They don't work for
 - abstract class
 - interface with more than one method

To Find All OdAccount with negative balance

```
//using regular/nested/inner class
class OdAccountNegativeBalanceMatcher implements Matcher<BankAccount>{
    public boolean match(BankAccount account){
        return account instanceof OdAccount && account.getBalance()<0;
    }
}

//using anonymous class
Matcher<BankAccount> matcher = new implements Matcher<BankAccount>{
    public boolean match(BankAccount account){
        return account instanceof OdAccount && account.getBalance()<0;
    }
}
```

- we have a lot of redundant information
- What is the really important piece of information in this class

Java 8 lambda intends to simply the anonymous model

- The idea of lambda is to remove those piece that can be inferred based on LHS
- Since we are creating a Matcher<BankAccount> we can understand
 - we want to implement the interface Matcher<BankAccount>
 - we will need "new" object
 - the interface will have a public boolean match method
 - matcher will take a Object of type BankAccount

```
Matcher<BankAccount> matcher = new implements Matcher<BankAccount>{
    public boolean match(BankAccount account) {
        return account instanceof OdAccount && account.getBalance()<0;
    }
}
```

simplified form

```
Matcher<BankAccount> matcher = (account) ->{
    return account instanceof OdAccount && account.getBalance()<0;
};
```

If your lambda expression has a single return statement

- You can remove
 - function block markers
 - return keyword
 - semicolon of return statement
- It becomes a lambda expression

```
Matcher<BankAccount> matcher = account -> account instanceof OdAccount &&
account.getBalance()<0
```


Exception Handling

Wednesday, February 15, 2023 2:48 PM

What is an Exception?

- unexpected situation
- not supposed to happen
- Undesirable
 - But expected
 - So that we can be ready to handle it
- In programming it is an error situation that should be tackled

Two parts of exception Handling

1. The Source of Exception
 - The point/object where the error occurs
 2. The Handler
 - One who is supposed to handle the problem
- In exception handling the two parties are expected to be different and NOT same.

Three Steps of exception Handling

1. source recognizes the error condition
 - generally with some if check
 - example
 - if authentication fails
 - if amount<0
 - if amount>balance
2. sending a signal to the handler (informing the handler)
 - What can be a signal
 - generally a invalid value can be returned as signal
 - example returning
 - ◆ false
 - ◆ null
 - ◆ Double.NaN
 - ◆ BankingStatus.invalidCredentials
 - setting a invalid flag
 - ◆ isValid()

3. Handling the exception by the handler
 - a. done using a if check again

Problem

1. We return invalid value for error

- same return statement returns a value and also error
- how do I know if the return is error or valid value?
- what if I don't have any value that can act as signal
 - suppose a function is expected to return int
 - How will it indicate a failure?
- since we can return anything there is no standard value for error
- What if a function doesn't have a return type
 - constructor?
 - how do we indicate error?
- When returning error we often lose the detailed error information
 - example if withdraw fails for insufficient balance we can say insufficientbalance but not how much insufficient
 - in balance transfer we may get the answer invalidaccountnumber but how do we which of the two account was invalid

2. return can return to immediate caller function

- actual error source and handler may be quite apart in function call chain.
- the source return value to its caller which returns to its caller and so on
 - authenticate returns to BankAccount.withdraw
 - BankAccount.withdraw returns to Bank.withdraw
 - Bank.withdraw returns to atm.withdrawMenu
 - atm.withdrawMenu returns to atm.mainMenu
- You may have to write same return every where
- bank.withdraw() is neither the source nor target of error
 - it still needs to check if authentication fails I have forwarded the error

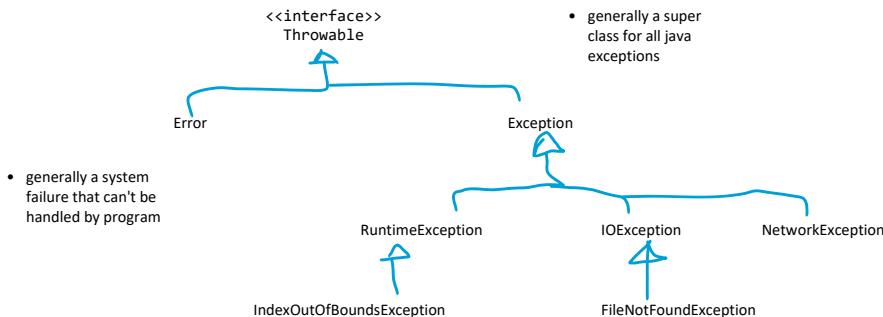
An ATM model

```
ATM
  • start()
    ○ displayMainMenu() <-- we want to handle errors here
      ■ displayWithdrawMenu()
        □ bank.withdraw()
          ◆ bank.getAccount()
          ◆ BankAccount.withdraw()
            ○ BankAccount.authenticate() <-- suppose authentication fails here
```

Object Oriented Exception Handling

1. What is an Exception

- Exception is an Object that inherits Throwable or one of its sub class
- We can create custom exception by inheriting from any of the given points
- Generally it is recommended to extend RuntimeException or its subclass



2. Raising the signal

- source raises the signal by "throwing" and exception object
- Note we don't return, we throw
- Unlike return we don't have to always specify what we throw
- we can return a different type and throw a different type
- When we throw it propagates till somebody handles it
 - intermediates may not have any role to play.

3. Handle the exception using try-catch

```
try{
    someFunctionThatMayThrow();

    //if you reach here, no error occurred
} catch( ExceptionType1 ex){
    //your each here is try block throws ExceptionType1
}

catch( ExceptionType2 ex2){

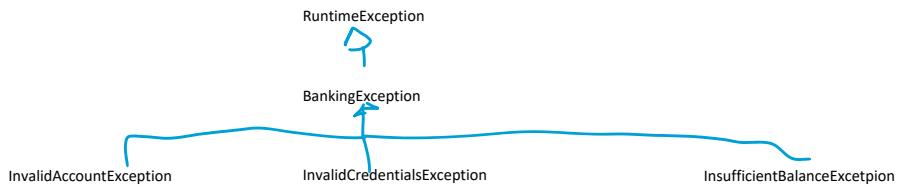
    //you reach ehre if try block throws Exception Type 2
    //once you reach here we believe you handled the exception even if you
    did nothing
}
```

Note

- we don't need try-catch to throw
 - if there is no try-catch exception will reach JVM and it will crash
- try indicates a block that contains a code that may throw exception
 - It indicates our readiness to handle exception
 - A try may be followed by one or more catch blocks
 - each catch block handles an exception of its own type

We can have User Defined Exception

- We can create our own class that sub classes any existing class in the hierarchy like
 - Exception
 - **RuntimeException**
 - **Recommended**
 - IOException
 - ...
- Because Exception is a class we can include any information in this class
 - Exception in Java can't be generic!!!
 - Example
 - account number
 - deficit balance
- We can create our own exception hierarchy



Assignment

- Create Exception Hierarchy
- Implement Exception in the banking classes
- Update Unit tests

Object Oriented Design

Friday, February 17, 2023 8:06 AM

Object

- Models a domain entity
 - OrganizationManagement
 - Employee
 - Project
 - Department
 - SalarySlip
 - Resource
 - HospitalManagement
 - Doctor
 - Patient
 - Department
 - Treatment
- A unit of responsibility
 - We define what responsibility this object has
 - What are the properties Object needs to fulfil its responsibility
 - Helps us create reusable domain objects

- It should be replaceable
 - when the requirement changes, an object can be replaced with another of same or similar (compatible) type.
 - When a tungsten bulb fuses
 - we can replace it with another
 - tungsten bulb
 - cfl
 - led
 - smart light
 - ... (who knows the future)
 - The only criteria
 - It should fit in a similar socket.

Class

- class → classification
 - A description for the object
 - and its hierarchy

How does static polymorphism or compile time polymorphism.

- this kind of polymorphism is achieved using method overloading.
 - creating different methods with same name and different parameters
 - parameters may differ in
 - number
 - type

Is it really polymorphism?

- conceptually two overloaded methods are as different as two methods with different names
- It is just a convenience in some basic cases.
- compiler invokes the right version by looking at the exact arguments
- If it fails to detect the argument, it will result in compile time error.

```
class ParkerPen{

    public void use(Hand hand){
        return "writing";
    }

    public void use(Pocket pocket){
        return "status";
    }
}
```

Encapsulation

- process of creating a unit of responsibility
- It defines an object model by
 - binding its state and behavior together to create a working and reliable model
 - we may use scope rules or convention to prevent access to private components
 - The idea is to define how state and behaviors are interconnected.

Polymorphism

- different objects
- behave differently (internal implementation)
 - how is encapsulated
- in the same context (external usage)
 - is external functionality or interface
- Polymorphism is achieved using Inheritance/interface

Inheritance/Interface

- They allow us to create class hierarchy
- Is a type of relationship
- Objects of same hierarchy should be polymorphically compatible to each other
 - similar to each other
- Is a tool to achieve polymorphic design.

Aside — How C++ translated overloading to C?

```
//C++ code
int divide(int a, int b){
}

double divide(double a, double b){

}

//translated to C by name decoration/mangling
int divide_int_int(int a, int b){

}

double divide_double_double(double a, double b){}
```

```

    public void use(Pocket pocket){
        return "status";
    }

}

Object getHandOrPocket(int type){
    //returned at runtime
    if(type==0)
        return new Hand();
    else
        return new Pocket();
}

double divide_double_double(double a, double b){
}

```

```

@Test
public void testInPocket(){

    Object obj = getHandOrPocket(1);

    var pen=new ParkerPen();
    var result = pen.use(obj);
    assertEquals("status", result);
}

```

will this pass or fail?

- neither.
- compiler fails

Error: no overload of use accepts Object type

- overloading works on reference and NOT on actual object
- It is compile time and NOT runtime.

```

@Test
public void testInPocketCorrectWork(){

    Object obj = getHandOrPocket(1);

    var pen=new ParkerPen();
    String result="";

    if(obj instanceof Pocket)
        result = pen.use((Pocket) obj);
    else
        result = pen.use((Hand) obj);

    assertEquals("status", result);
}

```

- These are two different sockets
 - one for pocket
 - one for hand
- hand and pocket can't be polymorphic substitute for each other.

Problem with Overloading

- An overloaded set of methods belongs to same object
 - The object can't replace itself.
- violates Open Close Principle
 - If you have three overloads today, you may have a fourth one tomorrow
 - Adding next overload will force to break OCP.
- violates Single Responsibility Principle
 - each overload represents a different responsibility
 - More responsible you are more likely to change in future

Assignment → create the parker pen model with hand and pocket using true polymorphism (override)

- HINT: ParkerPen should have a single use method

```

interface PenConnect{
    String penUse();
}

class ParkerPen {
    public void use(PenConnect connect){
        String connect.penUse();
    }
}

```

```

PenConnect getHandOrPocket(int type){

    if(type==0)
        return new Hand();
    else
        return new Pocket();
}

```

```
        }

    public void use(PenConnect connect){
        String connect.penUse();
    }
}

class Hand implements PenConnect{
    public String penUse(){ return "write"; }
}

class Pocket implements PenConnect{
    public String penUse(){ return "status"; }
}

else
    return new Pocket();
}

@Test
public void testInPocket(){
    PenConnect c = getHandOrPocket(1);
    ParkerPen pen=new ParkerPen();
    var result= pen.use(c);
    assertEquals("status", result);
}
```

Exception Part 2

Friday, February 17, 2023 10:52 AM

Exception Stack Trace

```
Exception in thread "main" java.lang.RuntimeException: Invalid Credentials
  at in.conceptarchitect.banking.BankAccount.authenticate(BankAccount.java:81)
  at in.conceptarchitect.banking.BankAccount.withdraw(BankAccount.java:67)
  at in.conceptarchitect.banking.Bank.transfer(Bank.java:76)
  at in.conceptarchitect.banking.ATM.doTransfer(ATM.java:97)
  at in.conceptarchitect.banking.ATM.mainMenu(ATM.java:34)
  at in.conceptarchitect.banking.ATM.start(ATM.java:21)
  at testapp04.bankaccountdemo.Program.main(Program.java:18)
```

Source.

- any of the below point till "main" can be the handler

- How return works
 - at each intermediate function we need to check for error and return
 - sometimes we need to convert error from one type to another based on return type of intermediate function

Exception automatically propagates

- from point of origin (source) till someone catches it
- if no one catches it
 - It reaches JVM
 - JVM displays the stack trace message (above)
 - JVM terminates the current thread

To Handle This exception

- we can write a try-catch around any desired point in the stack trace.
- We have chosen the main menu.

User defined Exceptions for Specific Business Cases

```
2
3 public class InvalidCredentialsException extends RuntimeException {
4
5     public InvalidCredentialsException() {
6         super("Invalid Credentials");
7     }
8 }
```

Handling specific and generic Exceptions

- for a single try block you can have multiple catch
- catch block can be written in any order if they are not class and sub class
- In case you want to catch both super class and sub class exceptions you must first handle specific (sub class exception) and then superclass.

```
}catch(RuntimeException ex) {
    System.out.println("Error:"+ex.getMessage());
}catch(InvalidCredentialsException ex) {
    Sys*Unreachable catch block for InvalidCredentialsException. It is already handled by the catch block for RuntimeException
}
}
System.
while(choice...)
```

- The correct approach is

Exception Hierarchy

- We can have our own exception hierarchy
- RuntimeException
 - BankingException (should include account number as details)
 - InvalidCredentialsException (should include consecutive wrong attempts ← assignment)
 - InvalidAmountException
 - InvalidAccountException
 - InsufficientBalanceException (should include deficit)

```
3 public class BankingException extends RuntimeException{
4
5     int accountNumber;
6
7
8     public int getAccountNumber() {
9         return accountNumber;
10    }
11
12    public BankingException(int accountNumber) {
13        this(accountNumber,"Banking Exception");
14        // TODO Auto-generated constructor stub
15    }
16}
```

Unit Testing Exception

- there can be different approaches

Option #1. Write a try-catch (Manual Process)

```
@Test
public void withdrawShouldFailForNegativeAmount() {
    try{
        account.withdraw(-1, password);
        fail("Expected InvalidDenominationException wasn't thrown");
    }catch(InvalidDenominationException ex) {
    }
}
```

Note

- We expect exception should be thrown
- You fail when exception is not thrown
 - we indicate this by writing fail in try block
- we may also assert against exception message in catch block

Option#2 Add expected property to @Test annotation

```
@Test(expected = InvalidCredentialsException.class)
public void withdrawShouldFailForInvalidPassword() {
    account.withdraw(1, "invalid password");
}
```

- Here test passes if expected exception is thrown
- Test fails if
 - no exception is thrown
 - some other exception is thrown

Option #3 Use assertThrows function

- Assert Throws function takes and Runnable interface

```
interface Runnable{
    void run();
}

• Now we can write the actual code we want to run inside execute function and pass to assertThrows
• If expected exception was thrown from execute the test will pass else fail

• This syntax works best with Lambda function
• we can write our logic as
```

```
assertThrows( () ->{
    //my logic here
});
```

```
@Test
public void withdrawShouldFailForInsufficientBalance() {
    assertThrows(InsufficientBalanceException.class, ()->{
        //your code here
        account.withdraw(amount+1, password);
    });
}
```

Advantage

- assertThrows **returns** the exception object if expected exception was thrown
 - Note the method catches and returns the object. It doesn't throw
- Now we can write asserts against exception properties

```

@Test
public void withdrawShouldFailForInsufficientBalance() {
    var ex= assertThrows(InsufficientBalanceException.class, ()->{
        //your code here
        account.withdraw(amount+1, password);
    });

    //you can also validate against exception properties
    assertEquals(5001, ex.getDeficit(),0.0);
}

}

```

finally block

- each try block can be followed by 0 or more catch block and 0 or 1 finally block
- A try must have atleast one catch or one finally following it
- finally if present must be the last block after all catch

```

try{
}catch(Ex1 ex1){
}catch(Ex2 ex2){
}finally{
}

or

try{
}catch(Ex1 ex){
}

or

try{
}finally{
}

```

Not Allowed is

```

try{
}

```

Why finally

- in a method with a try and catch block we can't be sure if
 1. whole try block will execute
 - we may get exception
 2. catch block will execute
 - we may not get exception
 - we may get some other not matching exception
 3. code after try-catch
 - you may get an exception for which catch block is not present
- if we need to ensure that some code (Generally cleanup code) must execute in all situations we put them in finally block
- A finally block executes
 1. In case of no exception -> soon after try block
 - try -> finally -> rest of the code -> exit from method with return
 2. In case of an exception that is caught
 - it jumps out of try,
 - executes catch
 - executes finally after catch
 - executes rest of the code
 - try -> catch -> finally -> rest of the code -> exit from method with return
 3. In case an exception is thrown but not caught
 - runs finally and then exits method with throw
 - try -> finally -> exit from method with throw
- finally is generally used for cleanup
 - close and open
 - file
 - connection

Checked vs Unchecked Exception

- Java supports two model of Exception Handling

Checked Exceptions

- All exceptions that are inherited from Exception super class except those inherited from RuntimeException
- For all checked exception Java compiler verifies if you have written a try-catch for it or not
 - This is to ensure that you have handled the exception
- Any method that may throw an exception must do one of these two things
 - write and try-catch
 - declares 'throws' on the method signature
- Now any method that calls a method that 'throws' must again do the same thing.

```
public void authenticate(String password) {
    Encrypt en=new Encrypt();
    if(!this.password.equals(en.encrypt(password)))
        //throw new RuntimeException("Invalid Credentials");
        throw new InvalidCredentialsException(accountNumber);
}

public void changePassword(String oldPassword, String newPassword) {
    authenticate(oldPassword);
}

public void authenticate(String password) throws InvalidCredentialsException {
    Encrypt en=new Encrypt();
    if(!this.password.equals(en.encrypt(password)))
        //throw new RuntimeException("Invalid Credentials");
        throw new InvalidCredentialsException(accountNumber);
}
```

`public void withdraw(double amount, String password) {`

`authenticate(password);`

Unhandled exception type InvalidCredentialsException
2 quick fixes available:
Add throws declaration
Surround with try/catch

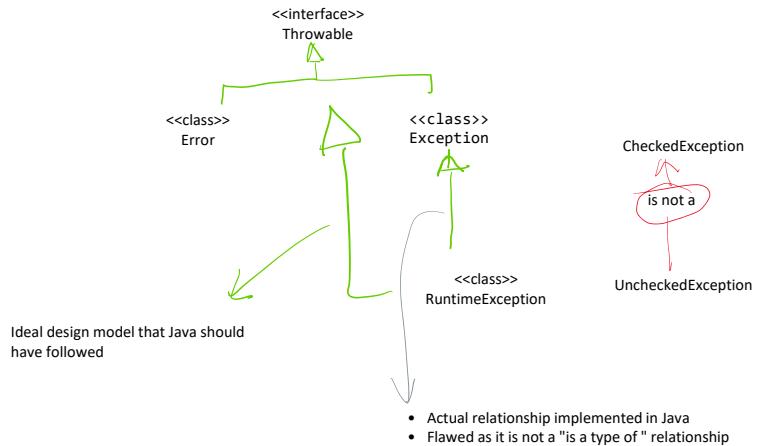
`option(accountNumber, amount-max);`

`balance-=amount;`

- indicates that this method may throw an exception that it doesn't catch
- This is like a "return" spec
- Any method that calls this again can either
 - provide try-catch
 - declare throws itself

Unchecked Exception

- those exceptions that are inherited from RuntimeException
- No throws clause is needed.
- Modern best practices guidelines strongly recommend to use Unchecked exceptions only
- Avoid using checked exceptions.



Converting Checked Exception to Unchecked Exception

- Checked exceptions are now considered as an anti-pattern.
- We should avoid creating new checked exceptions.
- But what to do with existing checked exceptions like InterruptedException or IOException or SQLException
 - Java initially created many checked exception

Solution

- Create a unchecked exception to replace the checked exception. You may include original exception as cause

```
class ThreadInterruptedException extends RuntimeException{
    Thread thread;
    public ThreadInterruptedException( Exception cause){
        thread=Thread.currentThread();
        super(cause);
    }
}
```

- We can replace Thread.sleep function with our own version of Sleep
 - Thread.sleep throws InterruptedException which is checked.
 - Our sleep will catch InterruptedException and throw the new Exception

```
class ThreadUtils{

    public static void sleep( int millisecond){
        try{
            Thread.sleep(millisecond);
        } catch(InterruptedException ex){
            throw new ThreadInterruptedException(ex);
        }
    }
}
```

Now if client is sure this exception will not occur, they may ignore it without a try or throws

```
void clientThatKnowsNoExceptionWillBeThrown(){

    try{
        Thread.sleep(2000);
    }catch(InterruptedException ex){
        //will never reach here
    }

    ThreadUtils.sleep(2000);
}
```

If client actually has a need of catching the exception it is still easy

```
void clientThatNeedsToCatchException(){

    try{
        ThreadUtils.sleep(2000);

    }catch(ThreadInterruptedException ex){
        //if you need the original exception
        var original = ex.getCause();
    }
}
```

Should I always handle exception or may return a sensible default

- If we expect an error and we have a sensible default for those scenario, we can write code accordingly

```
public String readString(String prompt) {
    try {
        System.out.print(prompt);
        return reader.readLine();
    }catch(IOException ex) {
        return "";
    }
}
```

- Many standard Libraries provide two variants of a function
 - One that throws error on invalid value
 - One that may return a sensible default on invalid value
 - generally these methods have "try" prefix
 - example
 - tryConvert
 - ◆ if conversion fails return a null

```
class DateUtils{

    public static Date tryParse(String value){

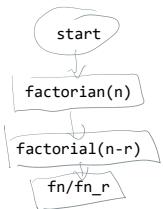
        try{
            return Date.parse(value);
        } catch(Exception ex){
            return new Date(); //today's date
        }
    }
}
```

Multi-threading

Friday, February 17, 2023 1:50 PM

```
int permutation(int n, int r){  
    int fn = factorial(n);  
    int fn_r=factorial(n-r);  
    return fn/fn_r;  
}
```

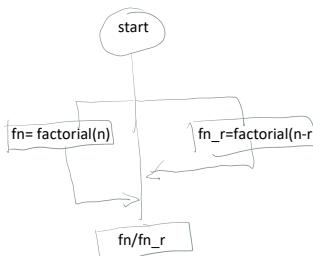
Current Model



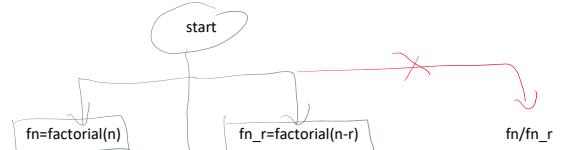
Why have we calculated factorial(n) before calculating factorial(n-r)?

- we have to do something first
- why?
 - because we can do only one thing at a time.

Expected Model



Not everything can be independent



calculation depends on completion of the other two calculations.



- Each limb can be independent of each other
- If they can be independent few can be parallel/concurrent also
- Although the threads are independent, they are part of the same puppet
 - we need to synchronize their working failing which puppet may break.

How to work with multithreading (puppets) in Java

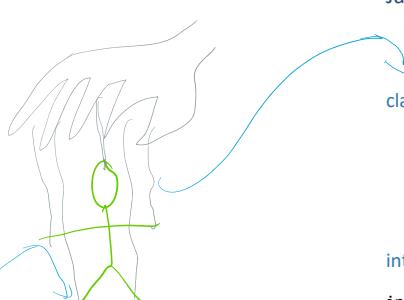
Puppet Show

Thread (Gray)

- controller for a portion of puppet (limb)
- makes a limb move or stop.

Limb That is being controlled (Green)

- A portion of the puppet that should work independently
- It could be anything based on current puppet show
 - hand of a person
 - wing of a bird
 - sword of a warrior
 - ...



Java or any other programming language

class java.lang.Thread

- controls portion of a program (sub task)
- starts or stops the working of a sub task

interface java.lang.Runnable

```
interface Runnable{  
    void run();  
}
```

Why interface?

- It represents a **user defined logic** that should run independently.
- Remember, as a thread controller we don't know what we are controlling. It could be different logic like
 - wings of bird

1. Every application is either multi-threaded or single-threaded.

- there is no un-threaded application
- every application we wrote so far had a thread which was running main function
- We can check out this thread's details.

The screenshot shows an IDE interface with three tabs at the top: BankingException.java, BankAccount.java, and Program.java. The Program.java tab is active, displaying the following code:

```

1 package testmt01.mainthread.app;
2
3 public class Program {
4
5     public static void main(String[] args) {
6
7         Thread currentThread = Thread.currentThread();
8
9         System.out.println(currentThread);
10    }
11 }
12
13 }
14

```

Below the code editor is a 'Console' window showing the output of the program:

```

<terminated> Program (11) [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (Feb 17,
Thread[main,5,main]

```

Handwritten annotations in blue ink are present in the screenshot:

- An arrow points from the word "name" to the word "main" in the output line.
- An arrow points from the word "priority" to the number "5" in the output line.
- An arrow points from the word "group" to the number "1" in the output line.

Thread elements

- Name
 - just for identification
- Group
 - to group several threads together
 - not much used
- priority
 - How important it is
 - varies in range 1-10 (highest)
 - default is 5
 - we can change these values

Creating New user defined thread

Friday, February 17, 2023 2:24 PM

- A thread is a controller for a Runnable
- There are couple of different ways we can create a thread

Approach #1 Create Runnable class

Step #1 : Create a class that implements Runnable

- The run() method will include the thread logic

Step #2 : Create the thread object and pass runnable object as argument

- Now the thread knows what to run when asked.
- Thread will not automatically start

Step #3:

- Start the thread by calling thread.start()

```
//Step #1
var printer=new ThreadInfoPrinter(); //this may run on a new thread

//Step #2
var userThread= new Thread(printer); //connect logic to a new thread

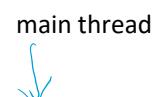
//Step#3
userThread.start(); //start the new thread and calls printer.run()

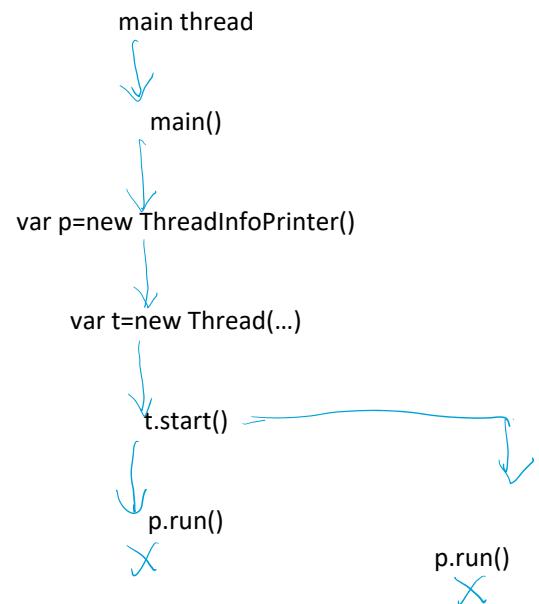
//printer is an ordinary object. It can be used even without new thread
printer.run(); //runs on the current thread printer.run()
```

```
terminator> program15, Java Application C:\program files\java\jrun
Thread[main,5,main] has id 1
Thread[Thread-0,5,main] has id 14
```

```
Thread[Thread-0,5,main] has id 14
Thread[main,5,main] has id 1
```

The Flow





Why second thread has id 14 and not 2?

- JVM runs several threads internally to manage the runtime
- Example
 - It runs garbage collector behind the scene
- It doesn't mean there are 13 threads in use.
 - Few threads are just reserved.

Creating Thread Alternative Models

Friday, February 17, 2023 2:40 PM

Approach #2 Extend Thread

- java.lang.Thread itself implements Runnable
- You may create a new class that extends Thread
- @Override the run()

Implements Runnable

```
Program.java  CountDownTask.java
3 public class CountDownTask implements Runnable {
4
5     @Override
6     public void run() {
7         int max=100;
8         System.out.printf("Thread %d starts\n",Thread.currentThread().getId());
9
10        while(max>=0) {
11            System.out.printf("Thread %d counts %d\n",Thread.currentThread().getId(),max);
12            max--;
13        }
14
15        System.out.printf("Thread %d stops\n",Thread.currentThread().getId());
16    }
17 }
```

Extends Thread

```
*CountDownLatch.java
1 package testmt01.mainthread.app;
2
3 public class CountDownLatch extends Thread {
4
5     @Override
6     public void run() {
7         int max=100;
8         System.out.printf("Thread %d starts\n",Thread.currentThread().getId());
9
10        while(max>=0) {
11            System.out.printf("Thread %d counts %d\n",Thread.currentThread().getId(),max);
12            max--;
13        }
14
15        System.out.printf("Thread %d stops\n",Thread.currentThread().getId());
16    }
17 }
```

```
Program.java  CountDownTask.java
5  public static void main(String[] args) {
6
7     //Step #1
8     var c1=new CountDownTask(); //this may run on a new thread
9     var c2=new CountDownTask();
10    var c3=new CountDownTask();
11
12    //Step #2
13    var t1= new Thread(c1); //connect logic to a new thread
14    var t2= new Thread(c2);
15    var t3= new Thread(c3);
16
17    //step#3
18    t1.start();
19    t2.start();
20    t3.start();
21
22 }
```

```
*CountDownLatch.java  *Program.java
4
5  public static void main(String[] args) {
6
7     //Step #1
8     var t1=new CountDownLatch(); //this may run on a new thread
9     var t2=new CountDownLatch();
10    var t3=new CountDownLatch();
11
12    //Step #2
13    t1.start();
14    t2.start();
15    t3.start();
16
17
18
19
20
21 }
```

```
class Thread implements Runnable{
    Runnable runnable;
    public Thread(Runnable r){
        this.runnable=r;
    }
    public Thread(){
        runnable=this;
    }
    public void run(){}
    public void start(){
        //starts new os thread
        runnable.run()
    }
}
```

```
interface Runnable{
```

```
    void run();  
}
```

Approach #3 Use Lambda function to implement runnable

```
public static void main(String[] args) {  
  
    var t1= new Thread(()->countDown()); //connect logic to a new thread  
    var t2=new Thread(()->countDown());  
    var t3=new Thread(()->countDown());  
  
    //step#3  
    t1.start();  
    t2.start();  
    t3.start();  
  
    System.out.println("end of program");  
}  
  
static void countDown() {  
    int max=100;  
    System.out.printf("Thread %d starts\n",Thread.currentThread().getId());  
  
    while(max>=0) {  
        System.out.printf("Thread %d counts %d\n",Thread.currentThread().getI  
        max--;  
    }  
  
    System.out.printf("Thread %d stops\n",Thread.currentThread().getId());  
}
```

```
class _____ implements Runnable{  
  
    public void run(){  
        countDown();  
    }  
}
```

How to pass parameter to a thread?

Friday, February 17, 2023 3:04 PM

- I may not always want to countdown from 10 or 100
- How do I pass parameter to CountDown Task
- Remember, run function can't take a parameter

When Implementing interface or extending class

```
class CountDownTask implements Runnable{
    int max;
    public CountDownTask(int max){
        this.max=max;
    }
    public void run(){
        int max=10;
        while(max>=0){
            ...
        }
    }
}
public static void main(String []args){
    var t1=new Thread( new CountDownTask(20));
    var t2 =new Thread( new CountDownTask(30));
    t1.start();
    t2.start();
}
```

When Using Lambda Function

```
static void countDown(int max){
    int max=10;
    while(max>=0){
        ...
    }
}
public static void main(String[] args){
    var t1= new Thread( () -> countDown(20));
    var t2= new Thread( () -> countDown(30));
    t1.start();
    t2.start();
}
```

Java Thread Model

Friday, February 17, 2023 3:12 PM

- In java application doesn't end when main() method ends
- main() method ends only main thread
- by default java application keeps running till at least one thread is running
- Ideally main thread should end after the child threads
 - if it ends before garbage collector may end up collecting the other thread memory
 - there is no one to control the child threads if we need.

How do I make main wait....

Approach #1 Let main do something even longer...

Reflection

Monday, February 20, 2023 8:31 AM

- Reflection is a library that helps you understand and utilize program structure dynamically.

```
class Employee{                                class Circle{  
    String employeeId;                      double radius;  
    String name;                            public double getPerimeter(){...}  
    Department department;                  public double getArea(){...}  
    Designation designation;  
    Salary salary;  
  
    public void work(){  
    }  
}
```

If everything is an object, is "class" too an object?

- Assume I have an object that represents a class like Circle

```
var cls ; //assume it to be a class object  
  
• what are the properties of this object  
  
class Class{  
    String name;  
  
    Method[] methods;  
    Constructor[] constructors;  
    Field [] fields;  
  
    Class superClass;  
    Class [] interfaces;  
    int scopes; //public, private, abstract, final  
}
```

But you can create an object of a class by calling new Class()

- We can't create an object of a class by creating

```
var c= new Class();  
• This syntax can't create a class. The right way to create a class is creating the source file
```

```
class Employee{  
    String employeeId;  
    String name;  
    Department department;  
    Designation designation;  
    Salary salary;  
  
    public void work(){  
    }  
    ...  
}
```

Everytime we create a class or an interface (or enum) java internally creates a 'Class' object for this class

- This object is pre-created
- There will be a single object created per Class

How to access this "Class" object

Option#1. If you can refer to the actual class

```
Class c1 = Employee.class; //gives access to the class object associated with Class
```

Option #2. If you have an access to any object of that class

```
Employee emp1 =new Employee(...);  
Employee emp2 = new Employee(...);  
  
Class c2 = emp1.getClass();  
Class c3= emp2.getClass();
```

important note

- all the references c1,c2,c3,c4 are referin to exact same instance of Class.
 - They are not creating new instance
- They will have the same hashCode

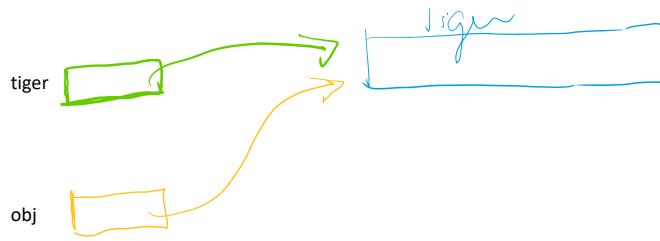
Option #3 If you know the class name (package qualified class name)

```
Class c4 = Class.forName( "in.conceptarchitect.ems.Employee");
```

What is the purpose of 'Class' class?

- Using 'Class' class helps us
 - get meta information about all the methods, fields and constructors present in a Class
 - This may help is creating dynamic documentation about the class.
- The Class also has a method called newInstance(), which can create an object of the current class using the default constructor.
- Class contains a list of constructors.
 - each constructor is an object of type Constructor
 - we can use parameterized cosntructor and call it's newInstance() method to create object using non-default constructor
- Class contains a list of methods
 - each method is an object of the Method
 - The object includes
 - name of the method
 - return type
 - parameters
 - scope
 - invoke
 - You can also invoke the method at runtime calling the method
- Class contains a list of fields
 - each field is of type Field
 - The object includes
 - name of field
 - type of field
 - scope
 - can be used to
 - get current value of the field
 - set new value in the field.





```
Tiger tiger =new Tiger();
```

```
Object obj= tiger;
```

```
obj = "Hello";
```

```
...
tiger.hunt() ; //works
obj.hunt(); //fails
```

IMPORTANT CONCERN!!!

- obj refers to actual Tiger object
- Tiger object has "hunt" method
- Why can't obj access hunt() method that actually exists on the object?
 - compiler is not sure if obj reference actually a Tiger
 - It plays safe and rejects those codes that may fail in future.

Why Can't we decide it at runtime?

- we can check if a method exist in the current object or not using Reflection
- Then we can invoke the method at runtime using Reflection.

```
@Test
public void aTigerCanHuntEvenWithObjectReference() throws NoSuchMethodException, SecurityException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException, ClassNotFoundException,
InstantiationException {
```

```
//Get class reference
Class cls = Class.forName("in.conceptarchitect.animals.mammals.Tiger");

//object obj=cls.newInstance();

Object obj = cls.getConstructor().newInstance();

//var result= obj.hunt();

//get the hunt method of the object
//if not found will throw NoSuchMethodException
Method hunt = obj.getClass().getMethod("hunt");

//invoke the method to get the result

var result = hunt.invoke(obj);

assertEquals("Tiger hunts in jungle", result);
```

- we can get the class information from a string that may come from user input
 - compiler may not be aware of this class
- If we have a class, we can create the object of that class using constructor
 - getConstructor()
 - takes parameter types
 - getConstructors()
 - get an array of all constructors
- we can get a access to the method of the class using
 - getMethod()
 - takes method name and required parameter types
 - or getMethods()
 - get an array of all the methods
- We can invoke the method by passing
 - obj that should invoke it
 - obj.hunt()
 - hunt.invoke(obj)
 - if hunt takes parameters they should come after obj

Annotation

- annotations are some meta information that can be associated with a
 - class
 - method
 - field
 - parameter
 - ...
- annotation generally work as a flag to mark some element for a special purpose
 - you can think of them as Marker interface

- Here we have conceptual translation from marker interface to an annotation
 - **NOTE: It will not work this way**
- **dd**
- An annotation can also include some additional information like
 - `@Test(expected=RuntimeException.class)`

How to create an annotation

- Annotation is introduced late in Java (6+)
- They reused the keyword "interface" with "@" prefix to define an annotation
 - It's more a patch than an interface.

```
@interface Pet{  
}  
}
```

- Now we can write

```
@Pet  
class Dog {  
}
```

How does annotation work?

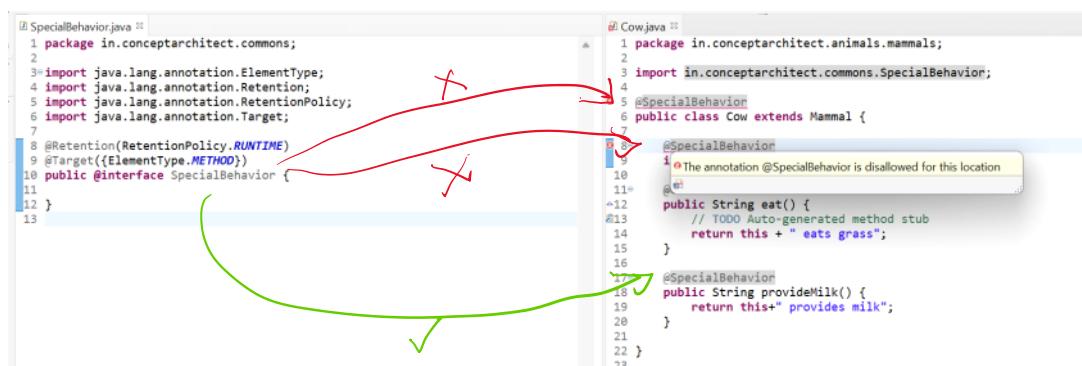
- annotation doesn't WORK on its own
- It's like a book sitting around.
 - it doesn't work on its own
- someone needs to open the book and read it
 - annotations are read and used using "reflection"

Use Case

- Every animal may have some special behaviors that can make it unique
 - Tiger → hunt()
 - Horse → ride()
 - Parrot → humanSpeak()
 - Dog → humanCompanion()
 - Cow → provideMilk
- How do I represent these special behaviors in programming?
 - No single interface can cover it all.
 - an animal may have more than one speciality.

Solution#1

- Let us create an annotation called `@SpecialBehavior`
- Let us tell java compiler that this annotation should be
 - available at runtime
 - applicable only on methods



- Now we have applied the annotation to selected methods
- But By default it does nothing.**
 - It must be checked and acted upon by some method
- If we don't have a method that uses annotation it is as good as some comment lying around

```
public static List<Method> findSpecialBehavior(Object obj) {
    // TODO Auto-generated method stub
    List<Method> methods= new ArrayList<>();

    var cls= obj.getClass();
    for(var method : cls.getMethods()) {
        if(method.isAnnotationPresent(SpecialBehavior.class))
            methods.add(method);
    }

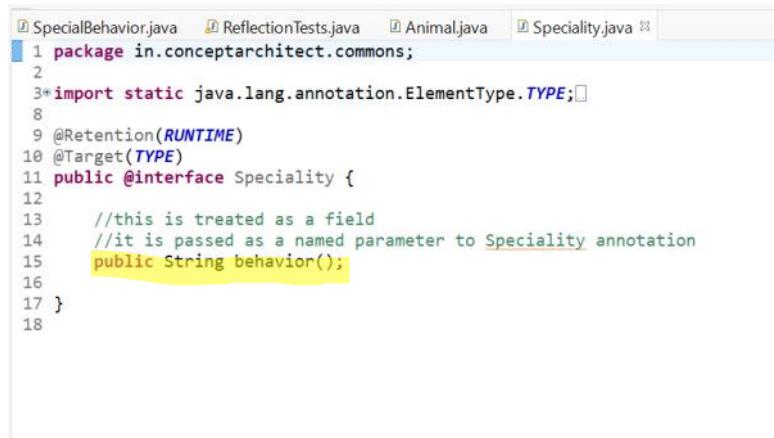
    return methods;
}
```

What if An animal can have a single @Speciality

- we can't prevent user from adding same annotation to multiple methods within the same class
- But we may create a design where
 - we can apply annotation to class rather than method
 - Pass the method name as a parameter to the annotation

How to add information to annotations?

- annotations are @interfaces
- interfaces can't have non-final field
- so we attach a property to annotation as a method
 - it looks like a method
 - but it acts like a field
- annotation is patchy design!!!



```
SpecialBehavior.java  ReflectionTests.java  Animal.java  Speciality.java
1 package in.conceptarchitect.commons;
2
3 import static java.lang.annotation.ElementType.TYPE;
4
5 @Retention(RUNTIME)
6 @Target(TYPE)
7 public @interface Speciality {
8
9     //this is treated as a field
10    //it is passed as a named parameter to Speciality annotation
11    public String behavior();
12 }
13
14 }
```

- Now we must pass the behavior property in annotation or give a default

```

6
7 @Speciality()
8 P The annotation @Speciality must define the attribute behavior
9
10= 1 quick fix available:
11  ↗ Add missing attributes
12
13
14 }
15

```

```

7 @Speciality(behavior = "humanSpeak")
8 public class Parrot extends Bird implements Pet{
9

```

- Note we can also have defaults for a property

```

public static Object invokeSpeciality(Object object, Object...params) {
    // TODO Auto-generated method stub
    var cls=object.getClass();
    var speciality= cls.getAnnotation(Speciality.class);

    if(speciality==null)
        throw new MethodInvocationException(cls.getSimpleName()+" has no speciality");

    var behaviorName=speciality.behavior(); //note invoked like method

    Method method=null;
    for(var m : cls.getMethods()) {
        if(m.getName().equals(behaviorName)) {
            method=m;
            break;
        }
    }

    if(method!=null) {
        try {
            return method.invoke(object, params);
        } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {
            // TODO Auto-generated catch block
            throw new MethodInvocationException( e.getMessage(),e);
        }
    } else {
        throw new MethodInvocationException(cls.getSimpleName()+" has no speciality "+behaviorName,
                new NoSuchMethodException(behaviorName));
    }
}

```

Assignment

Monday, February 20, 2023 11:13 AM

Create your own Unit Testing Framework

- The framework will run on Console
- Your test case file should resemble the standard test files
 - @Test
 - @Ignore
 - @Before
- you should run the test in the given way

```
class MyTest{  
    @Before()  
    public void setup(){  
    }  
    @Test  
    public void test1(){  
        assertEquals(2,3);  
    }  
    @Ignore @Test  
    public void test2(){  
        fail("This should be ignored");  
    }  
    @Test  
    public void test3(){  
        throw new RuntimeException("This should result in error");  
    }  
    @Test  
    public void test4(){  
        assertThrows( RuntimeException.class, ()-> throw new  
        RuntimeException("This should pass"));  
    }  
}
```

Interfaces and Lambdas

Wednesday, February 22, 2023 8:04 AM

```
interface IAction{  
    void start(Configuratin config, bool throwException);  
    void end();  
}
```

```
//implementation option #1 --> named class  
  
class MyAction1 implements IAction{  
    @Override public void start(Configuratin config, bool throwException){...}  
    @Override public void end(){...}  
}  
  
//implementation option #2 --> anonymous class (can be done only inside a  
method)  
  
var action2 = new IAction(){  
    @Override public void start(Configuratin config, bool throwException){...}  
    @Override public void end(){...}  
}
```

Lambdas work only with Functional interface

- A Functional interface is an interface that has single abstract method
- It may be optionally decorated with @FunctionalInterface annotation
 - introduced in java 8
 - if applied and you add more than one method you get compile time error

```
@FunctionalInterface  
interface ICommand{  
  
    void execute(Configuratin config, bool throwException);  
}
```

Implementing the Interface

```
class MyCommand1 implements ICommand{  
    @Override public void execute(Configuratin config, bool  
        throwException){...}  
}  
var command1=new MyCommand1();  
  
//implementation option #2 --> anonymous class (can be done only inside a  
method)

- The following code does two things
  1. Create a class (no name) that implements Iaction
  2. create an object of that class

  
var command2 = new ICommand(){  
    @Override public void start(Configuratin config, bool throwException)  
    {...}  
}  
  
//Implementation Option #3 --> Lambda Function  
  
ICommand command3 = (c,t) -> { ...}

- looks like we are just implementing a function that takes config and throwException and does something
- Actually this function has a name "execute" and is part of a class that implements ICommand
- How does it know it implements ICommand?
  - because we assign the lambda to ICommand
  - We can't assign it to "var" as it will fail to detect the method name of signature

  
var result = command3.execute( new Configuration(), false);
```

A lambda knows the signature best on where it is assigned

- assigned to a reference
- passed as method parameter
- Same lambda may actually mean different interfaces based on context

```
interface Command{  
    void execute();  
}  
  
interface Runnable{  
    void run();  
}
```

- For the implementation of the two interfaces using regular or anonymous classes it will be different
 - You need to implement different classes
 - You need to implement different method

- But for lambda it is going to be same

```
Command print = () -> System.out.println("Hello World");
```

- () -> System.out.println("Hello World") represents execute method

```
Runnable print = () -> System.out.println("Hello World");
```

- () -> System.out.println("Hello World"); represents run method

Function Reference Syntax.

- sometimes we may already have a function that has same signature as

Simple Calculator

Wednesday, February 22, 2023 8:32 AM

```
class SimpleCalculator{

    public int printAndReturn(int x, String opr, int y){

        int result;
        switch(opr){
            case "+": result=x+y; break;
            case "-": result=x-y; break;
            case "*": result=x*y; break;
            case "/": result=x/y; break;
            default: throw new InvalidArumentException("Unsupported operator :" +opr);
        }

        System.out.println(result);
        return result;
    }

}
```

```
var calc=new SimpleCalculator();

calc.printAndReturn(5,"+", 10);
calc.printAndReturns(20,"/",3);
calc.printAndReturn(50, "%",4);
```

Simple Lambda Example

Wednesday, February 22, 2023 8:22 AM

```
interface BinaryOperator{
    double calculate(double x,double y);
}

class Calculator{
    public double printAndReturn(double value1, BinaryOperator opr, double value2){
        var result= opr.calculate(value1,value2);
        System.out.println(result);
        return result;
    }
}

class CalculatorTests{
    int x=50;
    int y=15;
    Calculator calc=new Calculator();

    @Test
    public void canWorkWithPlusOperator(){
        var plus= new Plus();
        var result = calc.printAndReturn( x, plus, y);
        assertEquals(65, result);
    }

    @Test
    public void canWorkWithMinusOperator(){
        var result = calc.printAndReturn(x, new Minus(), y);
        assertEquals(35, result);
    }

    @Test
    public void canWorkWithMultipyOperator(){
        // m is the object of a (new class that implements ) BinaryOperator
        var m = new BinaryOperator() { public double calculate(double x, double y){return x*y; }};
        assertEquals(750, calc.printAndReturn(x, m, y) );
    }

    @Test
    public void canCalculateModOperator(){
        assertEquals(5, calc.printAndReturn(x, new BinaryOperator(){public double calculate(double x, double y){return x%y; }}), y);
    }

    @Test
    public void canDivideUsingLambdaOperator(){
        //var d = new BinaryOperator() { public double calculate(double x, double y){return x/y; }};
        //BinaryOperator d = new BinaryOperator (){double calculate (double x, double y)->{return x/y; } };
        BinaryOperator d = ( x, y)>{return x/y; } ;
```

```

        assertEquals( 50.0/15, calc.printAndReturn(x, d, y);
    }

@Test
public void canFindModUsingLambdaOperator(){
    //var d = new BinaryOperator() { public double calculate(double x, double y){return x/y;}};

    //BinaryOperator d = new BinaryOperator (){double calculate (double x, double y){return x/y;} };
    //BinaryOperator d = ( x, y) ->{return x/y;} ;

    //if lambda contains a single statement you can drop {}, ; and return keyword
    //BinaryOpeator mod = (x,y) -> x%y ;

    assertEquals( 50.0/15, calc.printAndReturn(x, (a,b)->a%b , y);
}

```

Function Reference Model

- sometimes we may have an existing method that has similar signature as what we need
 - signature includes
 - parameters
 - return
 - It doesn't include
 - method name
 - scope
- In standard practice we can wrap the method in lambda

```

class Math{

    public static double pow(double x, double y){...}
}

```

```

@Test
public void canUseExistingMathPowUsingLambda(){

    var result = calc.printAndReturn( 5, (a,b) -> Math.pow(a,b) , 2);

    assert(25,result,0.01);
}

```

- Here we created a new class with a method that takes (a,b) ->
- The method called another method with exact same parameter Math.pow(a,b)

Java provides simpler syntax for lambda using Function Reference

```

@Test
public void useExistingMathPowUsingFunctionReference(){

    var result= calc.printAndReturn( 5, Math::pow , 2);
}

```

IMPORTANT

- Math.pow
 - although doesn't match BinaryOperator calculate method
 - it is a static function
 - Math doesn't implement interface
 - It is not called calculate
 - It matches the signature of lambda
 - lambda doesn't have explicit method name
 - it doesn't have explicit scopes like public or static
- My Lambda is just a wrapper to collect the value and pass to another method
 - In such cases we can use Method reference syntax

Method reference for non static method

```

class Permutation{

    public double get(double n, double r){

        ...
    }
}

```

```

    }
}

@Test
public void canCalculatePermutationUsingLambda(){

    var p =new Permutation();

    //var r = calc.printAndReturn(5, p, 3); //p doesn't implement interface
    var r = calc.printAndReturn(5, (x,y) -> p.get(x,y) , 3);

    ...
}

```

- here we are calling p.get(x,y) inside the calculate method of an anonymous class that implements BinaryOperator

```

@Test
public void canCalculatePermutationUsingFunctionReferece(){

    var p =new Permutation();

    //var r = calc.printAndReturn(5, p, 3); //p doesn't implement interface
    var r = calc.printAndReturn(5, p::get , 3);

    ...
}

```

Create A Calculator with following Design

Wednesday, February 22, 2023 9:35 AM

```
var calc= new Calculator();

calc.addOperator("+", (x,y) -> x+y);

calc.addOperator("-", (x,y) -> x-y);

calc.calculate(5, "+", 2); //should print 5+2=7

calc.calculate(5,"-",2); //should print 5-2=3

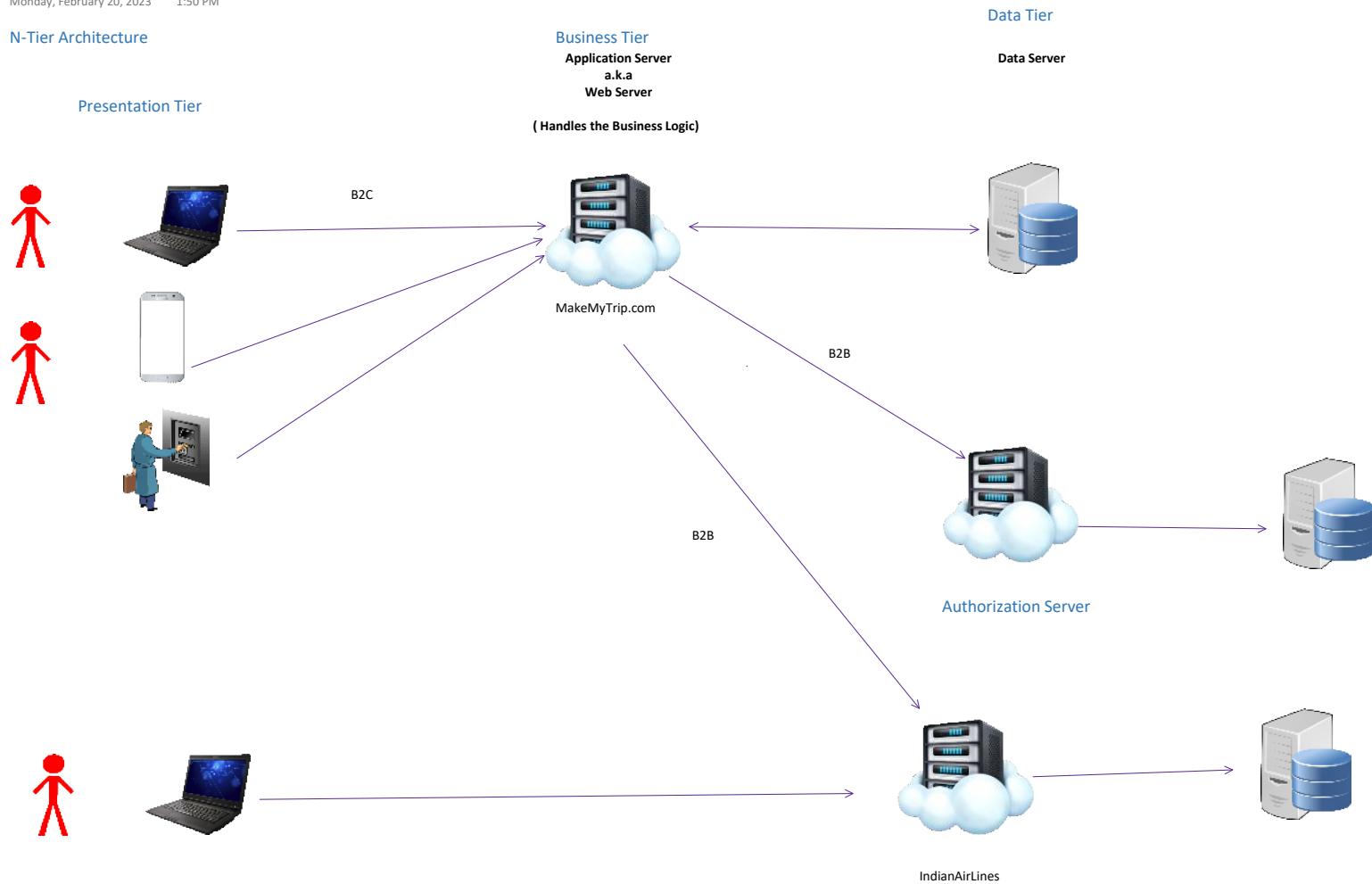
calc.caclulate(5,"*",2); //should throw InvalidArumentException
```

- Note
 - calculate is a void method and by default prints the result on console
 - But there should be a provision to
 - print the result somewhere else also like
 - gui
 - web page
 - save to a file
 - I may also want to see result either
 - full format
 - $2+3=5$
 - or raw format
 - 5
 - How do we unit test calculate?

A traditional distributed application

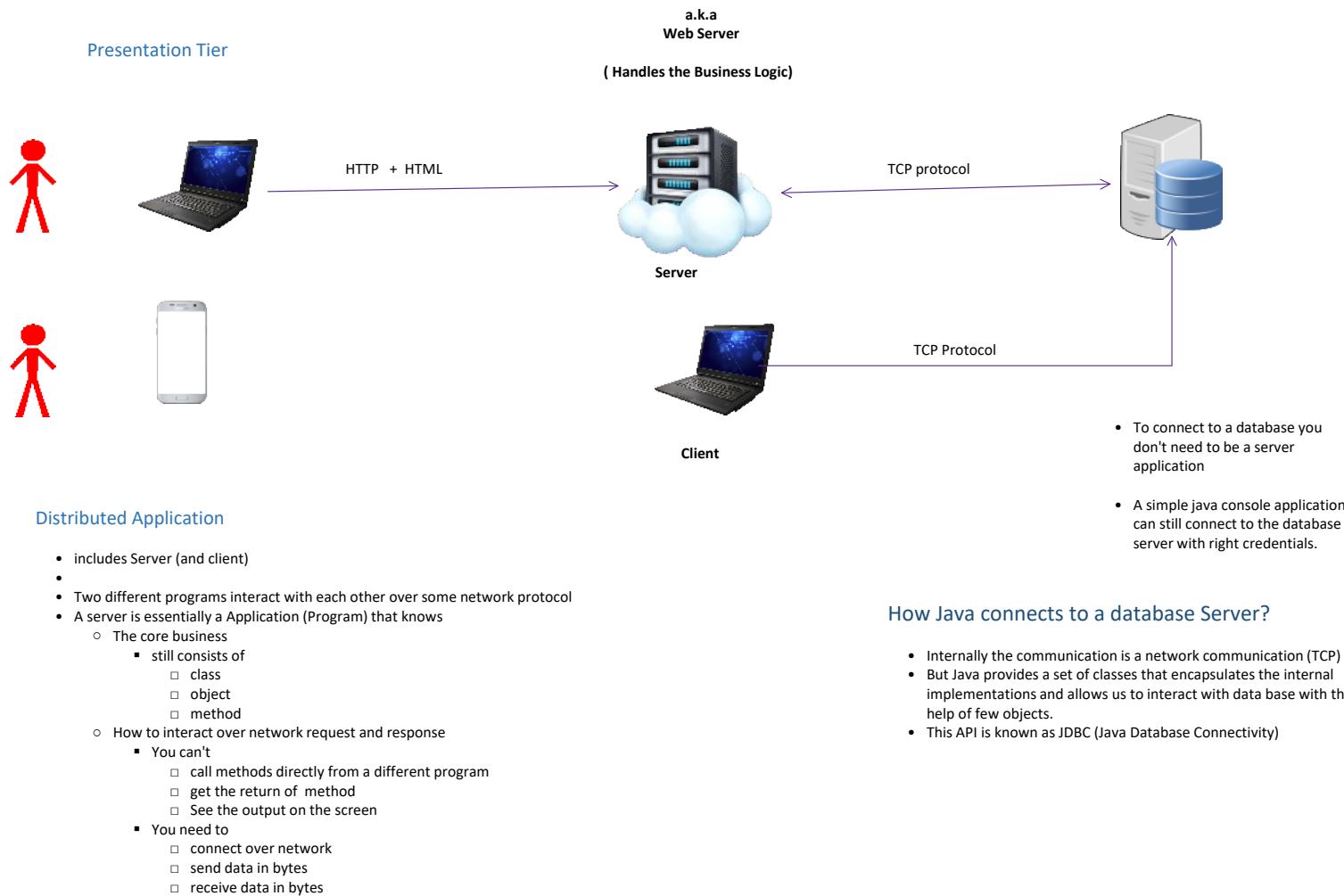
Monday, February 20, 2023 1:50 PM

N-Tier Architecture



Simple 3 Tier Architecture

Monday, February 20, 2023 2:07 PM



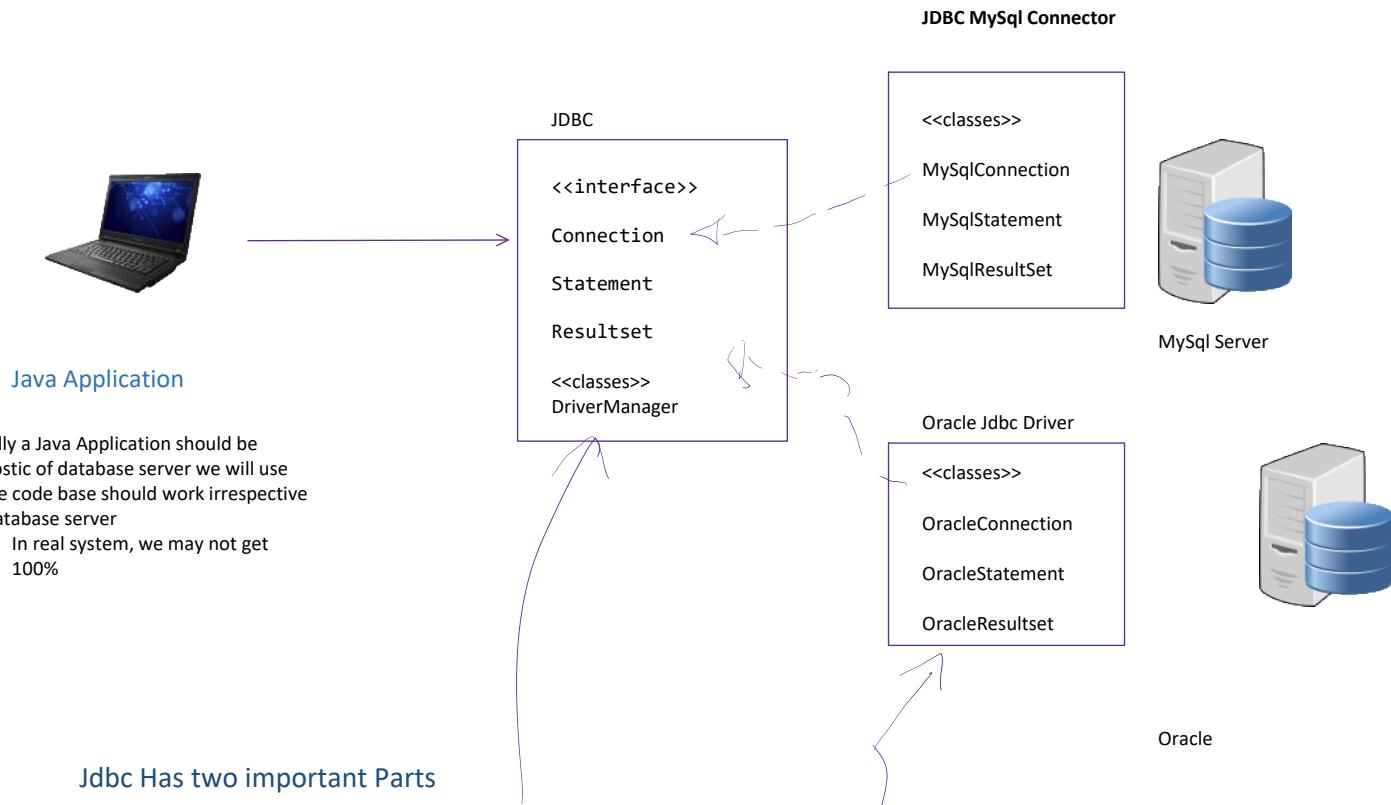
How Java connects to a database Server?

- Internally the communication is a network communication (TCP)
- But Java provides a set of classes that encapsulates the internal implementations and allows us to interact with data base with the help of few objects.
- This API is known as JDBC (Java Database Connectivity)

JDBC

Monday, February 20, 2023 2:16 PM

- A Java standard for a Java application to connect to any RDBMS system like
 - Oracle
 - MySQL
 - SqlServer
 - Postgres
 - ...



Jdbc Has two important Parts

1. An standard api that comes as part of core Java

- includes
 - interfaces
 - connection
 - how we connect to database
 - statement
 - query we need to send to database
 - ResultSet
 - Result returned by database
 - class
 - DriverManager
 - allows us to connect specific Driver to our application

2. Driver/Connector for specific database

- generally not part of core java library
- **provided by database vendors**
 - mysql connector should be obtained from mysql website
 - oracle connector should be obtained from oracle website
- The connector will be packaged as a jar file
- The jar contains implementations of JDBC interfaces specific to the underlying database
 - there may be implementation difference between how oracle and mysql connections are made

Getting Started with JDBC - MySql

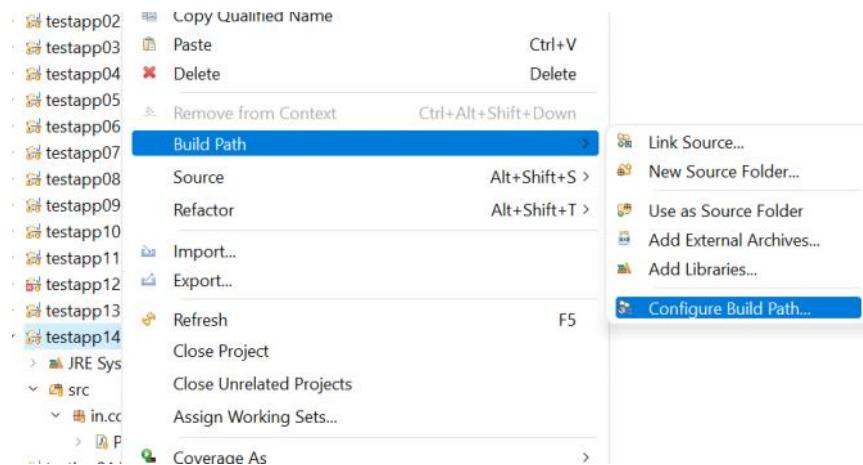
1. Install/procure the database Server

- we can install the server application
- we may subscribe to some cloud server that allows us to store data with them.
 - Example: elephantsql.com

2. Working with Java Program to Access the server.

- We need the connection string or credentials to connect to the database
- Generally it includes An URL containing
 - server address and port
 - database name
 - user name
 - password

3. Add the database sepecific connector jar to your build path



Important Steps to connect to database

1. Open the connection using DriverManager

```
String url = "jdbc:mysql://localhost/booksdb";
Connection connection = DriverManger.getConnection(url,username,password);
```

How will DriverManager know which Driver to user for jdbc:mysql?

- We may have included many (or none) of Jdbc Connector jars
- But how will DriverManager know which Driver class to create for opening current connection?

1. DriverManager creates a HashMap like below

```
HashMap<String, Driver> drivers;
```

Registration Option #1

- Jdbc Driver classes generally have a static block code that registers the Driver as soon as it is loaded in the memory
- Imagine a new Database company launched their connector

```
DriverManager.register( new MySqlDriver());
```

```

package com.newdb.drivers;

class NewDbConnector implements Driver{
    static{
        DriverManager.register(new NewDbConnector());
    }
    //Connector logic here
    ...
}

```

- Now in our client code, if we can load NewDbConnector in memory it will be automatically registered

```

class BookDbManager{

    public List<Book> getAllBooks(){

        var connection =
            DriverManager.getConnection("jdbc:newdb://localhost/bookdb");
    }

    public BookDbManager(){
        //Step 0
        Class.forName("com.newdb.drivers.NewDbConnector");
    }
}

```

Step #1 Should work if
DriverManager.register is called for
NewDbConnector

Step #0. We can load the Driver class in memory

- The driver class will automatically register itself to DriverManager

Registration Option #2 (Auto Registration in JDBC 4+)

- JDBC framework (DriverManager) automatically registers all the connector/driver class present in class path
- We no longer need to manually load the class in the memory for standard Java Runtime

```
Class.forName("com.newdb.drivers.NewDbConnector");
```

- Not needed for standard Java Runtime

Assignment

Monday, February 20, 2023 3:34 PM

- Create a BookDbManager class that supports basic CRUD functions
 - Get a list of all books
 - Add a new book
 - get a book by Id
 - update a book
 - delete a book

OO Patterns

Tuesday, February 21, 2023 10:20 AM

Constructor Problem

1. constructor has a completely meaningless name.
2. constructor is used with "new" and there is no "existing" keyword
 - a. It fails to differentiate between needing an object and needing a new object
3. constructor is non-polymorphic

Is Constructor class level (static) or object level (non static)

- is neither class level
 - no static keyword
 - has 'this'
 - can't be accessed using class reference
- nor object level
 - can't be accessed using object reference
 - `Circle c=new Circle();`
 - `Circle c2= c.Circle();`

Think!!!

- if it doesn't belong to class and doesn't belong to object what is it doing in the class?

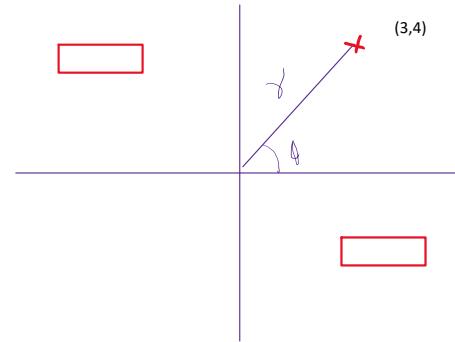
Has a completely Meaningless Name

- Class represents (describes) Object
- Constructor represents the creator of the Object
- A name that is meaningful for an object (or a product) doesn't become automatically meaningful for its creator also
 - The fact is, we have no control on the constructor name

Constructor means new

- It doesn't promote Object re-use

```
var p1= new Point(3,4); //it's a new point  
  
var p2= new Point(1,1); //it's also a new point  
  
var p3= new Point(3,4); //Is it a new Point?  
  
Point p1 = new Cartisan(3,4); //can't control the constructor name  
  
Point p2=new Polar(5,45); //can't control the constructor name
```



Geometrically each point is unique on a given plane

Constructors are non-polymorphic

- polymorphism works when a super class/interface reference refers to an object of the sub class
 - but when constructor is called, object doesn't exist yet to be referenced.
- In polymorphism when we call `shape.area()`, we may not be knowing which shape and how area will be calculated. It varies if shape is triangle or circle
 - In constructor you always know which constructor you are calling
 - `var x=new Circle();`
 - `var y=new Triangle();`
- constructor can't be overridden.
- constructor can't be part of the interface
 - can't implement DIP
- Dependency Inversion Principle
 - says know the abstract class / interface but not the concrete implementation
 - constructor forces you to know the concrete class
 - a constructor has same name as that of the concrete class.
- you can't replace a constructor with another to implement LSP
- Since it doesn't support DIS and LSP, it also tends to violate OCP
 - When we need to change the logic, we need

Constructors are NOT optional

- It's a language feature.
- It can't be removed.

```
class Point{  
    double x, y;  
  
    public Point(double x, double y){  
        this.x=x;  
        this.y=y;  
    }  
  
    public Point(double r, double theta){  
        var rad= theta*Math.pi/180;  
        x= r* Math.cos(rad);  
        y=r*Math.sin(rad);  
    }  
  
    public double getR(){  
        return Math.sqrt(x*x+y*y);  
    }  
  
    public double getTheta(){  
        var rad= Math.atan(y/x);  
        return rad*180/Math.pi;  
    }  
}
```

- Object creation always involves constructor call.

Solution —> We can't remove the constructor. But we can hide it.

- We can replace the constructor by hiding it behind some user defined method.
- Advantage
 - User defined method is user defined.
 - we can chose whatever name we like
 - we may choose if we want to create a new object or return an existing one
 - we may choose to create object of different types depending on the context.

```
class Point{
    double x, y;

    public Point(double x, double y){
        this.x=x;
        this.y=y;
    }

    public Point(double r, double theta){
        var rad= theta*Math.pi/180;
        x= r* Math.cos(rad);
        y=r*Math.sin(rad);
    }

    public double getR(){
        return Math.sqrt(x*x+y*y);
    }

    public double getTheta(){
        var rad= Math.atan(y/x);
        return rad*180/Math.pi;
    }
}

var p1= new Point(3,4); //x=3, y=4

var p2= new Point(5,45); //r=5, theta=45

class Point{
    double x, y;

    private Point(double x, double y){
        this.x=x;
        this.y=y;
    }

    public static Point getPoint(double x, double y){
        return getPoint(x,y);
    }

    public static Point getPolar(double r, double theta){
        var rad= theta*Math.pi/180;
        x= r* Math.cos(rad);
        y=r*Math.sin(rad);
        return getPoint(x,y);
    }

    public static Point getCartesian(double x, double y){
        return new Point(x,y);
    }

    public double getR(){
        return Math.sqrt(x*x+y*y);
    }

    public double getTheta(){
        var rad= Math.atan(y/x);
        return rad*180/Math.pi;
    }
}

var p1=Point.getCartesian(3,4);
var p2=Point.getPolar(5,45);
var p3=Point.getCartesian(3,4);

//is p1 == p3 //-->no
```

Re-using existing Objects

```
class Point{
    double x, y;

    private Point(double x, double y){
        this.x=x;
        this.y=y;
    }

    public static Point getPoint(double x, double y){
        return getPoint(x,y);
    }

    static List<Point> cache= new ArrayList<Point>();

    public static Point getPoint(double x, double y){

        for(var p : cache){
            if(p.x==x && p.y==y)

```

- p3 is re-using the same object as p1

```

        return p;
    }

    var p= new Point(x,y);
    cache.add(p);
    return p;
}

public double getR(){
    return Math.sqrt(x*x+y*y);
}

public double getTheta(){
    var rad= Math.atan(y/x);
    return rad*180/Math.pi;
}

}

var p1=Point.getCartisian(3,4);
var p2=Point.getPolar(5,45);
var p3=Point.getCartisian(3,4);
//is p1 == p3 //-->no

```

Factory Method

- Ordinary user defined methods that can replace a direct constructor call
 - factory can hide constructor not really replace it.
- Advantages
 - More meaningful name
 - can create or reuse object
 - can create objects of different types
 - can be mentioned as part of interface
 - can be overridden or replaced.

```
Connection con = DriverManager.getConnection(url, user,password); //which object is created?
```

```
Statement statement = con.createStatement();
```

Factory in a separate Class

```

class Point{
    double x, y;

    /*package*/ Point(double x, double y){
        this.x=x;
        this.y=y;
    }

    public double getR(){
        return Math.sqrt(x*x+y*y);
    }

    public double getTheta(){
        var rad= Math.atan(y/x);
        return rad*180/Math.pi;
    }
}

class PointFactory{

    public Point getPolar(double r, double theta){
        var rad= theta*Math.pi/180;
        x= r* Math.cos(rad);
        y=r*Math.sin(rad);
        return getPolar(x,y);
    }

    public Point getCartesian(double x, double y){
        return getPolar(x,y);
    }

    List<Point> cache= new ArrayList<Point>();

    static Point getPolar(double x, double y){
        if(cache.contains(x,y))
            return cache.get(x,y);
        else
    }
}

```

```

public double getTheta(){
    var rad= Math.atan(y/x);
    return rad*180/Math.pi;
}

var pf= new PointFactory();
var p1=pf.getCartisian(3,4);
var p2=pf.getPolar(5,45);
var p3=pf.getCartisian(3,4);

//is p1 == p3 //-->no
}

List<Point> cache= new ArrayList<Point>();

static Point getPoint(double x, double y){
    for(var p : cache){
        if(p.x==x && p.y==y)
            return p;
    }

    var p= new Point(x,y);
    cache.add(p);
    return p;
}

```

Unit Testing Jdbc Objects

Tuesday, February 21, 2023 11:37 AM

- Databases persists your records
- Each add test will add a new record in the database
 - that will impact total number of records in the database
 - assert will fail
- We MUST NOT run tests against production database
 - we must have separate test data base to run the code
- We must clear the rows or tables in @Before or @After

Refactoring Code

Tuesday, February 21, 2023 2:04 PM

```
public List<Book> getAllBooks(){  
    var books=new ArrayList<Book>();  
  
    Connection connection=null;  
    try {  
        connection=DriverManager.getConnection(url,userName,password);  
        var statement=connection.createStatement();  
        var rs= statement.executeQuery("select * from books");  
        while(rs.next()) {  
  
            var book=new Book();  
            book.setTitle(rs.getString("title"));  
            book.setAuthor(rs.getString("author"));  
            book.setPrice(rs.getInt("price"));  
            book.setRating(rs.getDouble("rating"));  
            book.setDescription(rs.getString("description"));  
            book.setTags(rs.getString("tags"));  
            book.setCover(rs.getString("cover"));  
            book.setId(rs.getString("isbn"));  
  
            books.add(book);  
        }  
  
        return books;  
    } catch(SQLException ex) {  
        throw new JdbcException(ex.getMessage(),ex);  
    } finally {  
        try {  
            if(connection!=null)  
                connection.close();  
        }catch(Exception ex) {  
            throw new JdbcException(ex.getMessage(),ex);  
        }  
    }  
}
```

```
public void addBook(Book book) {  
    Connection connection=null;  
    try {  
        connection=DriverManager.getConnection(url,userName,password);  
  
        var statement= connection.prepareStatement("insert into books "  
            + " (isbn,title,author,price,rating,cover,description,tags)"  
            + " values(?,?,?,?,?,?,?,?)");  
  
        statement.setString(1, book.getId());  
        statement.setString(2, book.getTitle());  
        statement.setString(3, book.getAuthor());  
        statement.setInt(4, book.getPrice());  
        statement.setDouble(5, book.getRating());  
        statement.setString(6, book.getCover());  
        statement.setString(7, book.getDescription());  
        statement.setString(8, book.getTags());  
  
        statement.executeUpdate();  
  
    } catch(SQLException ex) {  
        throw new JdbcException(ex.getMessage(),ex);  
    } finally {  
        try {  
            if(connection!=null)  
                connection.close();  
        }catch(Exception ex) {  
            throw new JdbcException(ex.getMessage(),ex);  
        }  
    }  
}
```

```
class DbManager{  
    public <X> X executeCommand(ConnectionCommand<X> x){  
  
        Connection connection=null;  
  
        try{  
            connection=DriverManager.getConnection(url,username,password);  
  
            //TODO: what?  
            x.execute(connection);  
  
        }catch(SQLException e){  
            throw new JdbcException(e.getMessage(),e);  
        } finally{  
            try{  
                if(connection!=null)  
                    connection.close();  
            }catch(SQLException e){  
                throw new JdbcException(e.getMessage(),e);  
            }  
        }  
  
        var statement=connection.createStatement();  
        var rs= statement.executeQuery("select * from books");  
        var books=new ArrayList<Book>();  
        while(rs.next()) {  
  
            var book=new Book();  
            book.setTitle(rs.getString("title"));  
            book.setAuthor(rs.getString("author"));  
            book.setPrice(rs.getInt("price"));  
            book.setRating(rs.getDouble("rating"));  
            book.setDescription(rs.getString("description"));  
            book.setTags(rs.getString("tags"));  
            book.setCover(rs.getString("cover"));  
            book.setId(rs.getString("isbn"));  
  
            books.add(book);  
        }  
  
        return books;  
    }
```

```
interface ConnectionCommand<T>{  
    T execute(Connection con);  
}
```

```
var statement= connection.prepareStatement("insert into books "  
    + " (isbn,title,author,price,rating,cover,description,tags)"  
    + " values(?,?,?,?,?,?,?,?)");  
  
statement.setString(1, book.getId());  
statement.setString(2, book.getTitle());  
statement.setString(3, book.getAuthor());  
statement.setInt(4, book.getPrice());  
statement.setDouble(5, book.getRating());  
statement.setString(6, book.getCover());  
statement.setString(7, book.getDescription());  
statement.setString(8, book.getTags());  
  
statement.executeUpdate();
```

```

Program.java
1 import in.conceptarchitect.bookmanagement.Book;
2 import in.conceptarchitect.bookmanagement.JdbcException;
3
4 public class DbManager {
5     String url;
6     String userName;
7     String password;
8     public DbManager(String url, String userName, String password) {
9         super();
10        this.url = url;
11        this.userName = userName;
12        this.password = password;
13    }
14    public <X> X executeCommand(ConnectionCommand<X> command) {
15        Connection connection=null;
16        try {
17            connection=DriverManager.getConnection(url,userName,password);
18            return command.execute(connection);
19        }catch(SQLException ex) {
20            throw new JdbcException(ex.getMessage(),ex);
21        }finally {
22            try {
23                if(connection!=null)
24                    connection.close();
25            }catch(Exception ex) {
26                throw new JdbcException(ex.getMessage(),ex);
27            }
28        }
29    }
30 }

```



```

BookDbManager.java
29
30
31
32
33
34 public List<Book> getAllBooks(){
35     return db.executeCommand(connection->{
36         var statement=connection.createStatement();
37         var rs= statement.executeQuery("select * from books");
38         var books=new ArrayList<Book>();
39         while(rs.next()) {
40
41             var book=new Book();
42             book.setTitle(rs.getString("title"));
43             book.setAuthor(rs.getString("author"));
44             book.setPrice(rs.getInt("price"));
45             book.setRating(rs.getDouble("rating"));
46             book.setDescription(rs.getString("description"));
47             book.setTags(rs.getString("tags"));
48             book.setCover(rs.getString("cover"));
49             book.setId(rs.getString("isbn"));
50
51             books.add(book);
52         }
53
54     });
55
56
57
58
59
60
61
62
63

```

Case Study: HDWallpapers

Wednesday, February 22, 2023 1:31 PM

Assume

- A website distributes free HD wallpapers
- Depends on Ad revenue source
 - Every Page will display an image
 - right click blocked for 20 seconds of ad time



What if we directly type the url for image?

<https://hdwallpapers.com/wallpapers/wallpaper14.jpg>

- Since HTTP is stateless, it doesn't know if you requested for html or not
- You can get the .jpg directly on the screen
 - NO HTML
 - NO JS
 - NO AD

How can HDWallpapers.com prevent such practice

HTTP Server Side

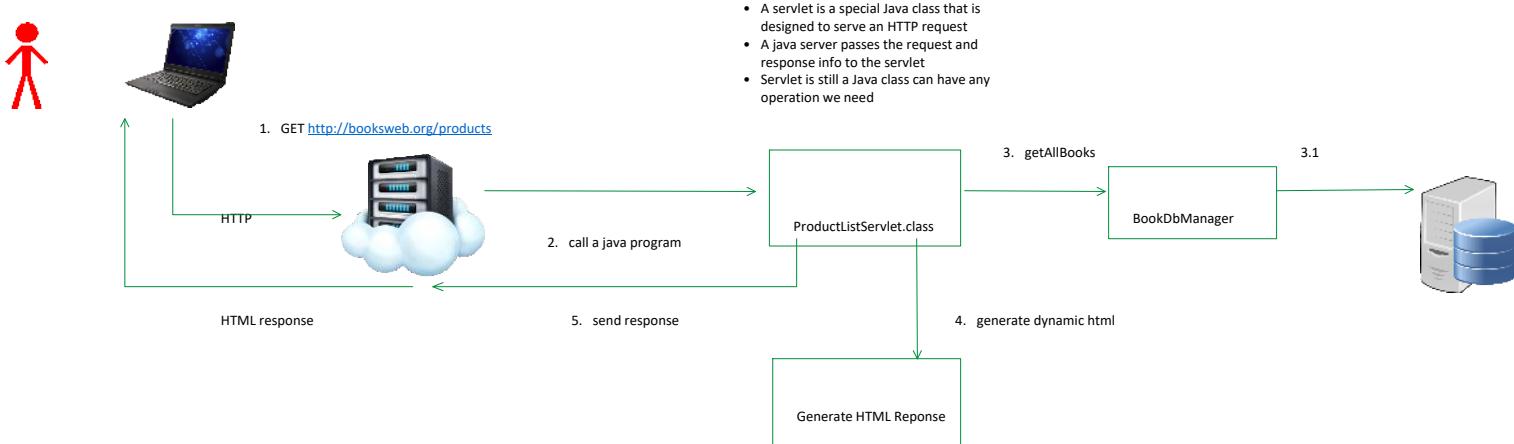
Wednesday, February 22, 2023 2:17 PM

A Traditional Web Server



Application Server

- An application server is a web server where contents are controlled and generated with the help of some application
 - data is fetched from some data server
 - pages are generated dynamically
 - pages are returned



Essential New Elements for Web Application

1. Web Server

- Not all webserver can serve Java Request
- We can use only those servers that know how to interact with Java
- Common Server
 - Apache Tomcat
 - IBM Websphere
 - Oracle Glassfish development
 - Weblogic

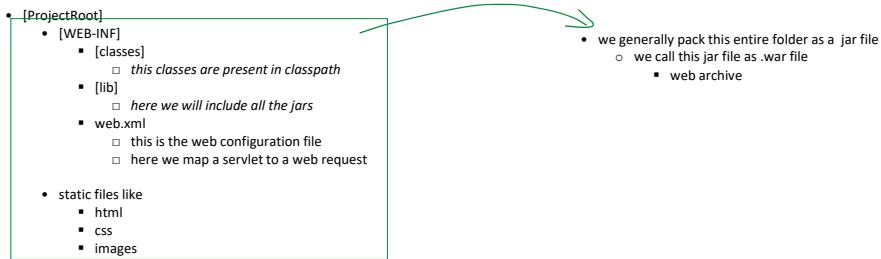
2. Our Servlet class can either

- implement
 - javax.servlet.Servlet interface
 - extend
 - javax.servlet.GenericServlet
 - or
 - javax.servlet.HttpServlet
- The servlet related library doesn't come out of box with java
 - It is generally available with the respective web server
 - just like JDBC driver is available with database vendor
 - Weblogic and apache may offer different jars

A WebApplication Structure

- As guided by Oracle





Deployment

- either copy .war file or the project folder in apache webapps directory
- Different servers may have different deployment technique

Implementing a Servlet

- A Servlet is a java class that
 - receives request in methods like
 - doGet
 - doPost
 - doDelete
 - doPut
 - doPatch
 - ...
 - the method is called with
 - HttpServletRequest
 - includes all request headers
 - HttpServletResponse
 - includes method to send response
- There will generally be one servlet per request type
 - BookListServlet --> /book/list
 - BookDetailsServlet --> /book/details
 - UserLoginServlet --> /user/logic
- We need to define a map to represent servlet and its mapping to a url
 - This is done in web.xml

Java Web Model

Monday, March 6, 2023 8:18 AM

Folder Structure



- On Application Root we keep static files like
 - html
 - css
 - js
 - These requests are served as
 - <http://books.org/index.html>
 - <http://books.org/images/logo.png>
- <root>/WEB-INF
 - Entire java code and dynamic app
 - This folder will NOT be served by server with normal URLs
 - <http://books.org/WEB-INF>HelloWorldServlet.java>
 - <http://books.org/WEB-INF/web.xml>

Servlet

- Classes that Java server will use to serve web request
- This class will have methods corresponding to all HTTP methods like
 - doGet
 - doPost
 - doDelete
 - doPut
 - ...
- When a request is received the appropriate method of servlet will be called with two parameters
 - ServletRequest request
 - contains all the http headers and other request parameters like sessions
 - ServletResponse
 - contains a writer that can send information back to the client
- Important Note
 - Servlets don't contain main method
 - Servlet runs inside an application server who is responsible for calling them

```
public class HelloWorldServlet extends HttpServlet{  
  
    @Override  
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException{  
  
        var writer = new PrintWriter( response.getWriter() );  
  
        writer.printf(format: "Hello World From Java Servlet. Today is %s", new Date());  
  
        writer.close();  
    }  
}
```

How server knows which servlet to call

- That is defined in a file called web.xml (traditionally)

```

1 <web-app>
2
3   <servlet>
4     <servlet-name>welcome</servlet-name>
5     <servlet-class>HelloWorldServlet</servlet-class>
6   </servlet>
7   <servlet>
8     <servlet-name>book-list</servlet-name>
9     <servlet-class>BookListServlet</servlet-class>
10  </servlet>
11  <servlet>
12    <servlet-name>book-details</servlet-name>
13    <servlet-class>BookDetailsServlet</servlet-class>
14  </servlet>
15
16  <servlet-mapping>
17    <servlet-name>welcome</servlet-name>
18    <url-pattern>/</url-pattern>
19  </servlet-mapping>
20
21  <servlet-mapping>
22    <servlet-name>book-list</servlet-name>
23    <url-pattern>/book/list</url-pattern>
24  </servlet-mapping>
25
26  <servlet-mapping>
27    <servlet-name>book-details</servlet-name>
28    <url-pattern>/book/details/*</url-pattern>
29  </servlet-mapping>
30
31 </web-app>

```

- we may have similar entries for different servlets that may serve our request.

Compilation and Deployment

1. Compile the code
2. Create a war file (renamed jar file)
3. Deploy on the server
4. launch the server

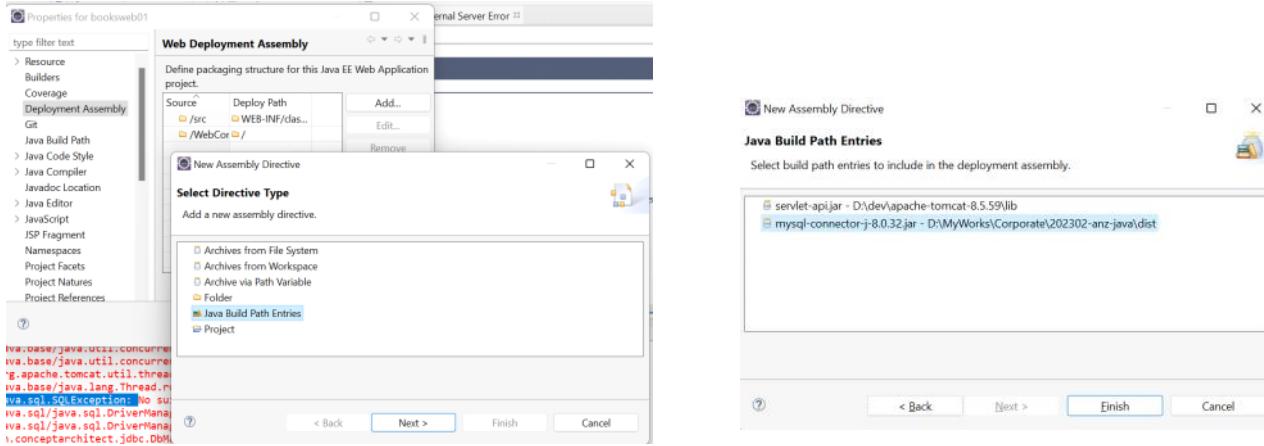
Eclipse Web Project Structure



- We keep our java source file here.
- Eclipse will keep the compiled code in WEB-INF\classes folder during deployment
 - It will not copy the source file
- Here WebContent folder is the project deployment root
- JavaResource is not a deployment component
- Eclipse Project structure may not work with other IDEs which may follow their own structure
- Eclipse runs the server and auto deploys server-api.jar

To Deploy 3rd Party Assemblies along with our web application

1. Right Click Eclipse Project



Eclipse Java Project doesn't support auto registration of Jdbc Drivers

- we must use approach #1

How do I display a complex HTML output

- currently we generated a HTML formatted output using `writer.printf()`

```
writer.printf("<h1>%s</h1>", book.getTitle());
writer.printf("<h2>by %s</h2>", book.getAuthor());
writer.printf("<ul>");
writer.printf("<li>Price: Rs %d</li>", book.getPrice());
writer.printf("<li>Rating: Rs %f / 5</li>", book.getRating());
```

- Here we inject html tags inside java source file
- String manipulation is difficult and time taking

- This may work for an extremely simple use case but will not be feasible for large and complex UI needing
 - CSS
 - Javascript
 - large html
- Most HTML designer won't be comfortable working this way

JSP

- JSP (Java Server Pages) looks like an HTML document with few special Java code
- They allow us to create major UI design in HTML syntax and then inject the required data
- Internally JSP is compiled back as a servlet
- JSP doesn't need to have web.xml entry

```
<%@page import="in.conceptarchitect.bookmanagement.Book"%>
<%@page import="java.util.List"%>
<%@page import="in.conceptarchitect.bookmanagement.BookDbManager"%>
<%@page import="in.conceptarchitect.jdbc.DbManager"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Our Favorite Books</title>
</head>
<body>
```

- JSP page header includes
 - JSP page specification
 - Import statement
- JSP mostly contains HTML content that may include
 - CSS
 - Java Script
- In JSP we may need to use Java Code that comes in two varieties

Java Code Block

- anything we write between `<% ... %>` is a Java Standard statement
- Since JSP is internally compiled as servlet all this information goes to "doGet" method

```
16  
17=<%  
18  
19     List<Book> books=null;  
20  
21     try{  
22         String url="jdbc:mysql://localhost/booksdb";  
23         String userName="root";  
24         String password="@DMIn.";  
25         Class.forName("com.mysql.jdbc.Driver");  
26         DbManager dbManager=new DbManager(url, userName, password);  
27         BookDbManager bookManager=new BookDbManager(dbManager);  
28  
29         books= bookManager.getAllBooks();  
30  
31     }catch(ClassNotFoundException ex){  
32     }  
33  
34  
35  
36 %>
```

- Note these codes don't have an output unless you use `response.getWriter()` here
- They don't present any output to the user

Writing a response (data) inside the HTML

- To inject content in the HTML that will become part of it we need to use `response.getWriter.append()`
 - A shortcut for this is `<%= ... %>`
 - Anything we write within the block is injected (added) to the output
-
- In this block we can only use expressions like
 - variable
 - expression
 - method call
 - here we can't use full statements like
 - try-catch
 - while
 - for
 - if



```
<% for(Book book : books){ %>
    <li><%=book.getTitle() %></li>
<%} %>
```

The diagram shows a green arrow pointing from the JSP code block to the corresponding Java code. The JSP code contains a loop and an expression. The Java code is annotated with a comment: `response.getWriter.append(book.getTitle());`

Problem with JSP

- Here we are injecting lot of java code inside HTML
- It is more difficult for a java developer to write code in jsp file than java (Servlet) file
- Not easy to maintain

Solution

- Create a servlet
 - Servlet will do the heavy java weight lifting
 - It should run the entire logic needed to fetch the data or do business processing

- loops
 - try-catch
- The servlet should process and generate data needed for the output
 - Now servlet should pass this data to some JSP
2. JSP should receive pre-processed data
- Its only job would be to add the data in HTML at the right place.

Step #1 Define Servlet

Project Management

Monday, March 6, 2023 9:24 AM

Problem with the current approach

1. Our application may depend on many external libraries like
 - a. jdbc driver
 - b. servlet api
 - c. other projects in our application
2. We may need to search for these libraries across the vendor websites and search for latest versions
 - a. We may not know which is latest version
 - b. Those libraries may have dependencies on yet another library
 - Library A —> LibraryB
 - Here Library A depends on Library B
 - Now latest version of Library A may not depend on latest version of Library B
 - ◆ latest version of library B may have some breaking changes that Library A can't use
 - c. Some libraries are needed only at development time
 - Unit testing library
 - servlet-api is needed at compile time but **provided** by server at run time
3. IDE based project structure may not be compatible with each other

Solution

- A project and dependency management toolkit that is independent of the IDE.
- Should allow us easy access to needed third party libraries.
- Should allow us to follow same structure across the IDE

Maven

1. Maven as Dependency Provider

- Maven is primarily a project dependency management tool or framework
- Maven central server (repository) hosts thousands of libraries needed for Java applications including
 - junit library
 - data base connectors
 - servlet-api
 - any third party library jar
 - your own library if you so wish.
- A developer side utility (application or tool) to get these components from the server

Maven Component Identification

- Each component (or library) present on Maven is uniquely identified using three keys
 - group id
 - generally it is like top level package or brand identification

- It can be anything developer likes but should be unique
- Generally a company may push all their library under one group id
 - example
 - ◆ org.springframework
 - ◆ org.apache
 - ◆ junit
 - ◆ org.hibernate
- artifact id
 - Id of current library or jar that we want to use
 - there may be multiple artifacts available under same group id
 - think of each artifact as one project (or jar or library) created by a brand
- version number (alphanumeric)
 - We may release multiple versions of the same jar over a period of time

2. Maven as Project Manager

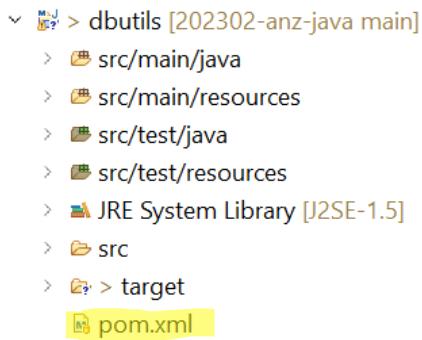
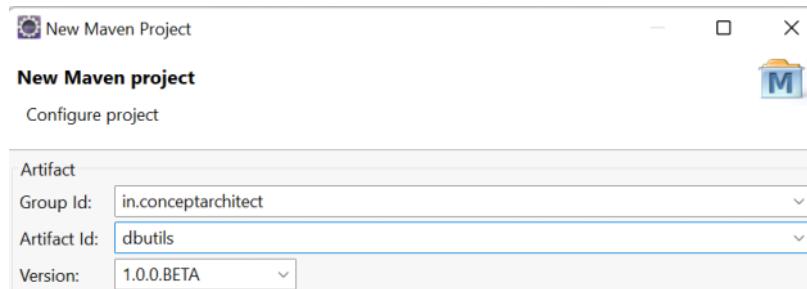
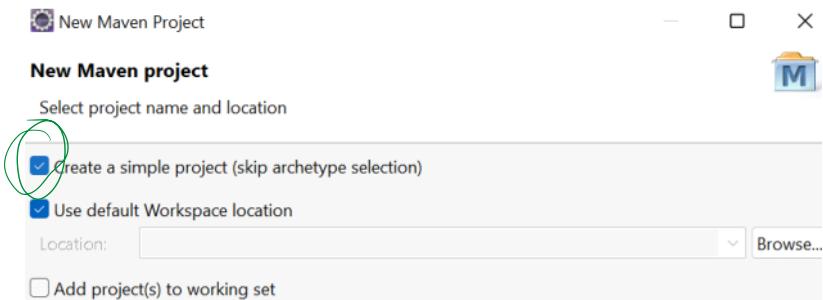
- Apart from the main goal of providing dependency maven is also a project manager
- It provides a basic structure for our project (something similar to how eclipse give, but quite different)
- Advantage of following maven structure is
 1. Maven projects can work across all IDEs
 2. It provides best practices guidelines
 3. **It allows us to**
 - Utilize existing maven component
 - Create our own maven component
- **Important parts of a maven project**
- **pom.xml**
 - It is the key configuration and dependency file for any maven project
 - It includes
 - groupId, artifactId, version-number of all our dependencies
 - It treats each maven project as maven component thus expecting you to provide for your own project
 - groupId
 - artifactId
 - versionnumber

Getting Maven

1. we may download and install maven on our machine to work with command line
2. Eclipse (and most IDEs) include their own maven utility

Starting with Maven

Monday, March 6, 2023 9:46 AM



- Here we are creating a "reusable library"
 - dbutils
- Later we may publish it to maven so that anyone can use it

- /src/main/java
 - Here we store our java source files needed to run the application
- /src/test/java
 - Here we should store our test files
- src/main/resources
 - non source file related to run of application
 - configuration files
 - json file
 - images
- src/test/resources
 - resources for test purpose only
- pom.xml
 - This is the most important part of maven

Maven Repository

- Maven Repository is a file structure that includes folders for
 - each fragment of group id
 - artifact id
 - version number
- Example for a project with
 - Identity
 - groupId: in.conceptarchitect
 - artifactId: dbutils
 - version: 1.0.0.BETA
 - folder structure
 - in
 - conceptarchitect

- ◆ dbutils
 - ◊ 1.0.0.0.BETA
 - dbutils.jar

- Maven maintains this folder structures at
 - maven repository (if the project is uploaded there)
 - you may have a free account with maven to upload your project in public
 - local maven repository per account on your PC
 - It is present under <user>/m2/repository

The screenshot shows a file explorer window with the following path: Vivek Dutta Mishra > .m2 > repository. The table lists four sub-folders:

	Name	Date modified	Type	Size
\$	in	11/3/2020 11:27 AM	File folder	
cd	info	1/7/2021 10:38 AM	File folder	
cd	io	1/7/2021 10:40 AM	File folder	
cd	jakarta	11/4/2020 12:42 AM	File folder	

- An organization may set their own maven respository that may be a sub set of approved components only

Making our component available locally

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a selected folder named "dbutils [202302-an]
- Run As:** A context menu is open, with "Run As" highlighted. A submenu shows various Maven goals: 1 Java Application, 2 Maven build, 3 Maven build..., 4 Maven clean, 5 Maven generate-sources, 6 Maven install (highlighted), and 7 Maven test.
- Console:** Shows the command: \$ mvn install
- File Explorer:** Shows the local Maven repository structure: Vivek Dutta Mishra > .m2 > repository > in > conceptarchitect > dbutils > 1.0.0.BETA. Inside this folder are three files: _remote.repositories, dbutils-1.0.0.BETA.jar, and dbutils-1.0.0.BETA.pom.

Handwritten annotations in green highlight the "Run As" menu path and the Maven goal "6 Maven install". Arrows point from the "Run As" menu to the "6 Maven install" option and from the "6 Maven install" option to the local repository files.

Assignment

Monday, March 6, 2023 11:19 AM

1. Create Maven Project : dbutils
 - includes generic library for db management
2. Create Maven Project: book-manager
 - includes bookmanager classes
 - includes tests
3. Create Maven Project: book-manager-console
 - includes a main function to display a list of books from production database

Web Application Using Maven

Monday, March 6, 2023 2:29 PM

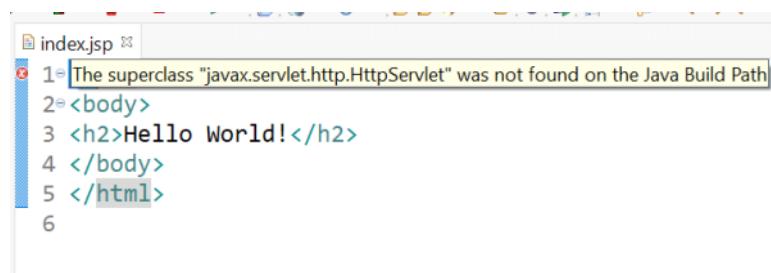
Maven Archetype

- Maven apart from providing project dependency also defines Project Structure (Archetype)
- A project structure pre-configures
 - folder structure
 - Some required assembly for the project
- By default we use a "simple archetype" which is good enough to create
 - class library
 - console application
- To create a web application we need special archetype



New Maven project

Select project name and location



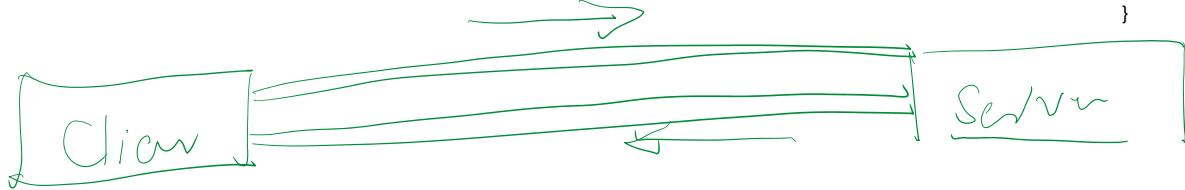
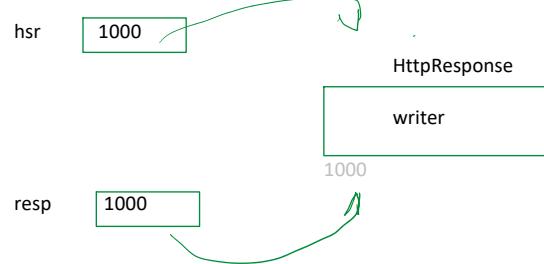
- Remember a JSP is internally a servlet
- And by default servlet-api.jar is NOT included.
- We must add it

How is response returned

Tuesday, March 7, 2023 8:08 AM

JavaHttpServer

```
class HttpServlet{  
    public void serve(Request req, Response resp){  
        HttpResponse hsr = new HttpResponse();  
        if(req.getMethod().equals("Get"))  
            doGet(req, hsr);  
    }  
  
    @Override  
    protected void doGet(HttpServletRequest req,  
                         HttpServletResponse resp) throws ServletException, IOException {  
        PrintWriter writer=new PrintWriter(resp.getWriter());  
        writer.printf("Hello World");  
    }  
}
```



```
void serve(){  
    var response=...  
    response.write(greet());  
}
```

Servlet - JSP communication

Tuesday, March 7, 2023 8:45 AM

Servlet

1. interacts with business layer to get the required information and do any java level processing
2. Any information that should be displayed in processed and added to the request using req.setAttribute
3. Now request is dispatched to the JSP

```
1  String url="http://localhost:8080/BookList";
2  DbManager dbManager= new DbManager(url, userName, password);
3  BookDbManager bookManager= new BookDbManager(dbManager);
4
5  List<Book> books = bookManager.getAllBooks();
6  //resp.getWriter().append("<h1>Total Books "+books.size());
7
8  //add any information needed by jsp to the request object
9  req.setAttribute("books", books);
10
11 req.getRequestDispatcher("/WEB-INF/jsp/book-list.jsp")
12 .forward(req,resp);
13 }
```

JSP

4. JSP receives the control along with request and response object
5. It can extract supplied information using request.getAttribute
6. Now it can display the data

```
1 <body>
2   <h1>Our Recommended Books</h1>
3   <%
4     List<Book> books=(List<Book>) request.getAttribute("books");
5   %>
6
7   <p>Total Books: <%=books.size()%>
8
9   <% for(Book book:books){ %>
10    <p><%=book.getTitle()%>
11   <% } %>
12
13 
```

JSTL (JSP Tag Language)

- A set of user defined Tags that can work with java data

```
5  <%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
6
7
8 <!DOCTYPE html>
9 <html>
10 <head>
11 <meta charset="ISO-8859-1">
12 <title>Our Recommended Books</title>
13 </head>
14 <body>
15   <h1>Our Recommended Books</h1>
16
17   <ul>
18
19     <c:forEach var="book" items="${books}" %>
20       <li>${book.title}</li>
21     </c:forEach>
22   </ul>
23
24
25 </body>
26 </html>
```

Problem with Servlet and JSP

Tuesday, March 7, 2023 9:30 AM

1. We typically need one Servlet per URL

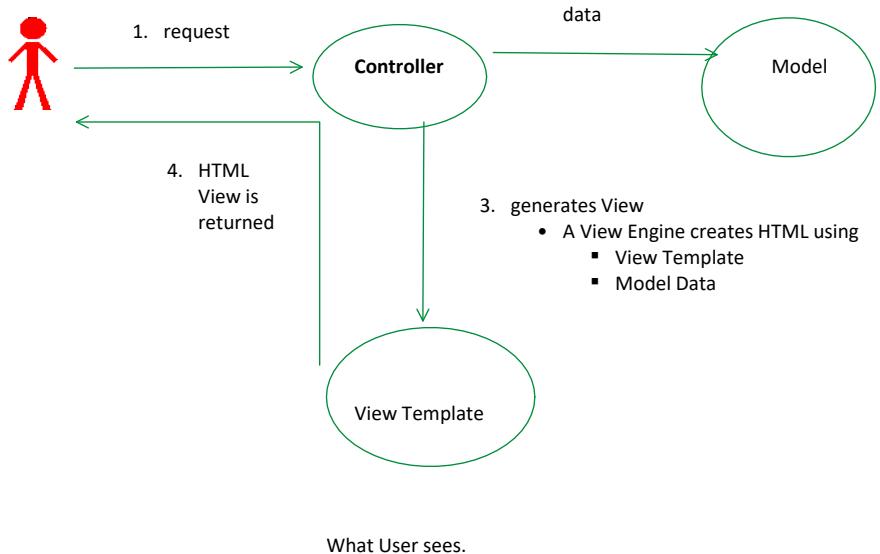
- It may handle different methods like GET,POST etc
- We may use same servlet for different urls like
 - <url-mapping path="user/*"/>
 - But generally this will violate SRP
- Each servlet requires two entries in web.xml
 - We have recently changed this approach (a little too late)

2. We have same initialization in multiple servlets

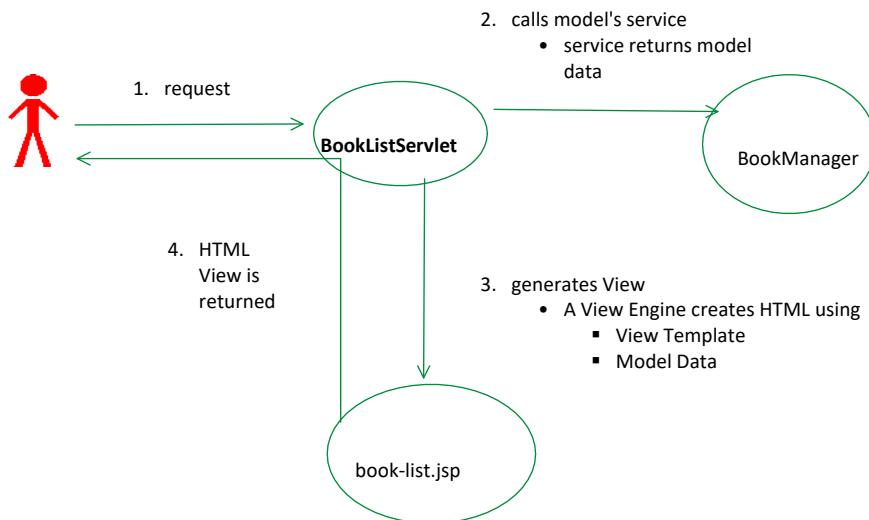
- Creating BookManager in
 - BookListServlet
 - BookDetailsServlet
 - BookAddServlet
 - BookDeleteServlet
- This violates DRY

MVC Pattern

Tuesday, March 7, 2023 9:34 AM



Option #1 Servlet as Controller and JSP as View



Model → Business Model

- includes
 - Business Information (data/entity)
 - Business Logic (service)
- Represents your entire Business

Core Goal

- Separate Model from the View

Why?

- Short Answer
 - A model can have multiple views
- Long Answer
 - Different Physical View
 - 19" or 5"
 - Architectural View
 - Desktop, Web, App?
 - Logical View
 - Pie chart or Bar Chart
 - Functional View
 - View or Edit?

- A Servlet can act as controller
 - User sends Request to Controller
- Servlet can create object of Business Service (BookManager) and get book list (model data)
- Servlet sends data to JSP View Engine along with jsp page name
- A JSP page can't be read by browser
 - browser doesn't understand
 - <% %>
 - <%= %>
 - \${}
 - JSTL
 - JSP View engine converts JSP to HTML by transforming JSP tags into plain text and injecting the data as per java/jstl logic
- The generated page is returned back to the user

Problem

- same as servlet problem discussed earlier

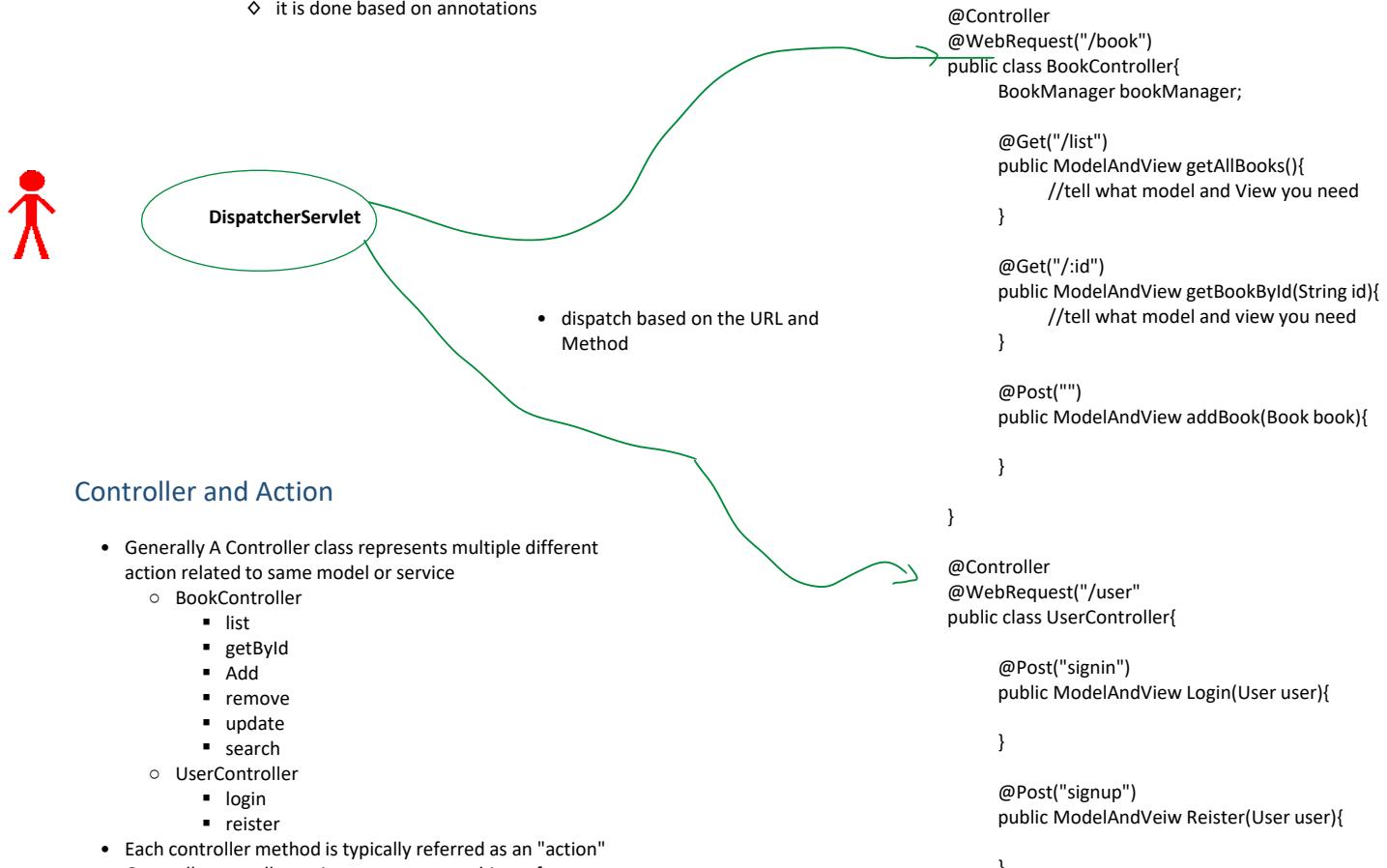
Option#2 Use some third party MVC frameworks like struts or SpringMVC

- These frameworks have separate controller classes
- One controller can have different methods to serve different URL
- URLs are mapped to methods using annotations rather than web.xml
 - Java copied this idea and now even servlet can be executed using annotation

- while this approach will reduce web.xml entries, it still needs multiple servlet creations
- Any Java based MVC framework still needs servlet which is essential building block for java web application
 - A controller can't receive request directly
- Third party frameworks like struts, jersey, SpringMVC follow a similar Dispatcher servlet pattern

Dispatcher Servlet (aka Front Controller) pattern

- The framework provides a single pre-created Servlet (generally called DispatcherServlet)
- We should configure this servlet to handle all user requests
 - single web.xml entry
 - All request reaches this dispatcher servlet
 - Are we violating SRP?
 - No
 - The job of this servlet is NOT to serve the request but
 - ◆ identify who (controller) can serve request
 - ◆ It is the only (single) responsibility
 - ◆ it is done based on annotations



- Note
 - Controller doesn't implement or extend any interface or super class
 - It is identified by annotation @Controller
 - Controller Actions don't have fixed names or signature
 - They are again identified by annotation that defines
 - HttpMethod
 - URL pattern
 - ◆ may include variables like :id

Distributed System Revisited

Tuesday, March 7, 2023 10:32 AM

N-Tier Architecture

Presentation Tier



B2C

HTTP → HTML
HTTP → JSON



HTTP → ???



HTTP → ??? → XML or JSON

Business Tier

Application Server
a.k.a
Web Server

(Handles the Business Logic)

MakeMyTrip.com

.

B2B

HTTP → ???

B2B



Authorization Server

.

B2B



Data Tier

Data Server

.

IndianAirLies

RESTful Service aka WebAPI

Tuesday, March 7, 2023 10:46 AM

- A HTTP based service that may return
 - JSON (mostly)
 - or other formats as per client requirement
 - XML
 - RSS
 - ATOM
 - VCARD
 - Plain Text
- An HTTP request that doesn't return HTML or SOAP
- Follows True HTTP Protocol Model
 - Proper Usage of
 - URL
 - Request Headers
 - Response Headers

What is a URL?

- Uniform (universal) resource locator

W3C Recommendation for a URL

- A URL should be
 - Short
 - Simple
 - Permanent
 - Identification of a **Resource**
- A URL shouldn't represent
 - Action
 - Technology
 - ...

Consider these two URLs?

<http://books.org/booksbyauthor.jsp?author=23912>

- long
- queryString looks ugly
 - ugly may not be BAD.
- .jsp binds this URL to Java technology
 - this is NOT permanent
 - If ever we need to move to another technology our business URLs shall change.

<http://books.org/books/by/23912>

- short
- clean
- permanent
- user friendly?
 - what does 23912 represent?
 - Jeffrey Archer
 - How do I know
 - How to find books by John Grisham?
 - What is his id

<http://books.org/books/by/jeffrey-archer>

- not so short
- user friendly
- Can you guess how to get all books by John Grisham?

URL represents Resource NOT Action

<http://books.org/authors> —> get all authors

class Author{

```
List<Book> books;
List<Testimonial> testimonials;
Blog blog;
List<Quote> quotes
```

}

<http://books.org/authors/vivek-dutta-mishra> —> get details (may be profile) of particular author

<http://books.org/authors/vivek-dutta-mishra/books> —> get books by the author

<http://books.org/authors/vivek-dutta-mishra/testimonials> —> get the testimonials

<http://books.org/authors/vivek-dutta-mishra/blogs> —>

<http://books.org/authors/vivek-dutta-mishra/quotes> —>

How do I represent actions?

GET all Books

- Actions comes from HTTP methods

GET <http://books.org/books/list>

- Here list can be considered as "action" or state
- We can also say

GET <http://books.org/books>

How Do I Add a New Book?

GET <http://books.org/books/add>

POST <http://books.org/books>

- here "add" is an action
- And this action doesn't match HTTP method "GET"
 - GET shouldn't add anything on server

- Here "POST" method represents "CREATE" which is what HTTP represents
- "add" is
 - redundant when used with "POST"
 - contradiction when used with "GET"

How do we Delete a particular book "brethren"

GET <http://books.org/books/delete/brethren>

DELETE <http://books.org/books/brethren>

- This will work but
 - It is a GET request
 - "delete" is an action word in the URL should be avoided

- Now the DELETE method represents delete action
- URL represents what to delete

URL shouldn't represent technology

GET <http://books.org/books/the-accursed-god?format=json>

GET <http://books.org/books/the-accursed-god>
ACCEPT: application/json

- we expect to get the book in JSON format
- when format is not specified we may assume HTML

Some examples of URL

<http://books.org/books>

- GET → get a list of all books
- POST → add a new book
- PUT → 405
- DELETE → 405

What an URL represents

- URL represent that state transfer (server to client or client to server) of a resource or its sub state
- It is known as
 - REpresentational State Transfer
 - REST

<http://books.org/books/the-accursed-god>

- GET → get details of book with friendly id "the-accursed-god"
- PUT → update the book details
- DELETE → delete the book with given id
- POST → 405

REpresentational State Transfer

<http://books.org/books/the-accursed-god/reviews>

- GET → get all reviews about the book
- POST → add a new review about the book
- DELETE → 405
- PUT → 405

Problem → Browser doesn't support PUT/DELETE/PATCH

- How will this kind of URL work
- This kind of URL can't work when browser makes the request on its own
- A typical web application still will have to follow
 - GET
 - POST
 - They will be forced to include "ACTION" in the URL

<http://books.org/books/the-accursed-god/reviews?min=3&max=4>

- GET → get all reviews about the book with rating range 3-4

<http://books.org/books/the-accursed-god/reviews/12>

- GET → get all review id 12 for book the-accursed-god

<http://books.org/books/the-accursed-god/reviews/12>

- GET —> get all review id 12 for book the-accursed-god
- POST —> 405
- DELETE —> delete this review
- PUT —> update this review

GET <http://books.org/books/the-accursed-god/cover>

ACCEPT: image/webp

- POST —>405
- PUT —> 405
- DELETE —> 405

<http://books.org/users>

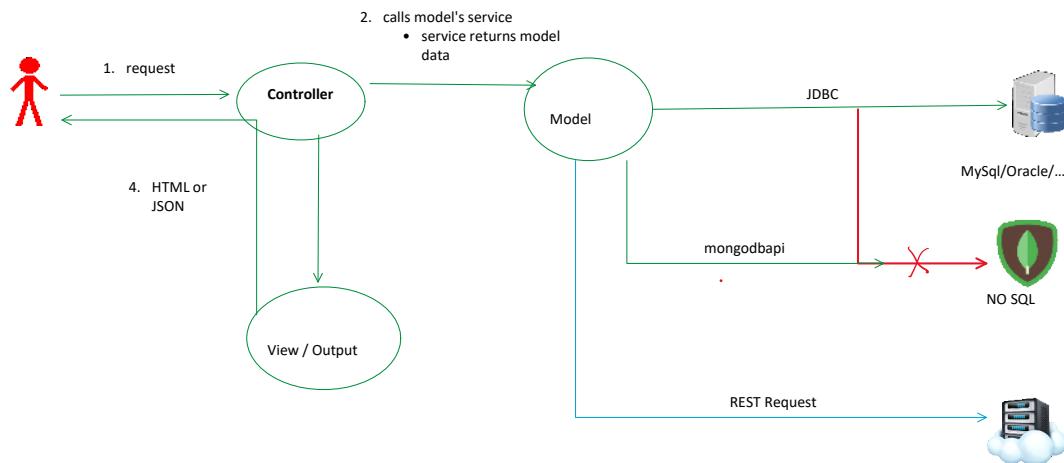
- POST —> create a new user
- PUT —> login a user
- PATCH —> update user details
- GET —> list all user (available only to ADMIN)

- POST
- They will be forced to include "ACTION" in the URL

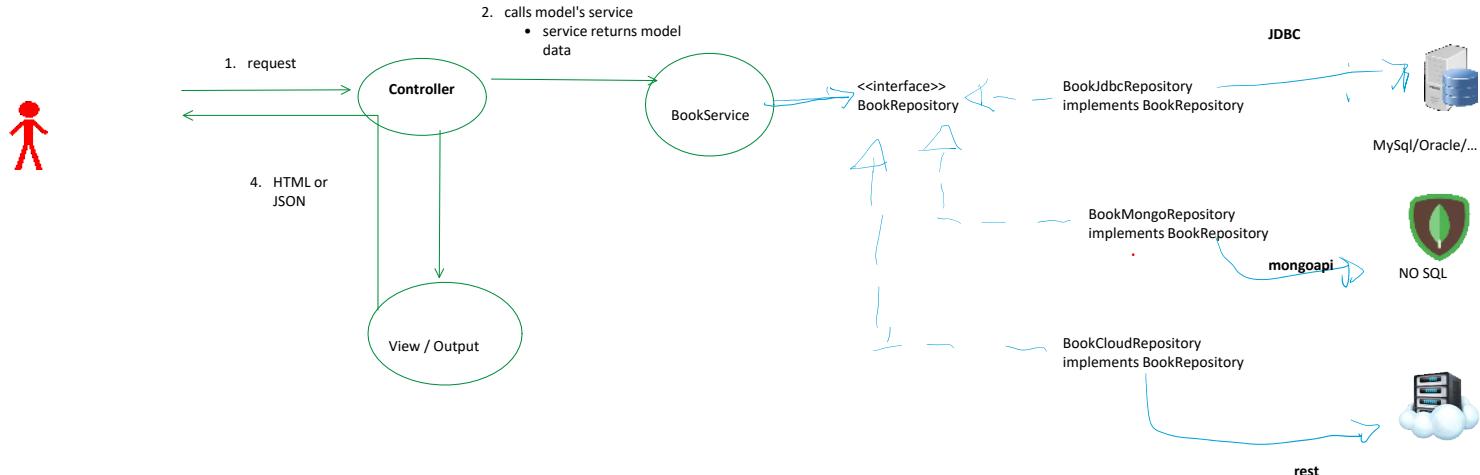
- REST style URL can work when
 - we use non-browser applications like
 - mobile app
 - desktop app with custom objects making http request
 - browser based application
 - when request is made via JavaScript Objects (AJAX)
 - instead of using form method and submit directly
 - we submit a request using Javascript method
 - JavaScript method can make all HTTP requests (PUT/DELETE/)

MVC based Architecture

Tuesday, March 7, 2023 11:30 AM



Repository Interface



```
class BookService{
    BookRepository repository; //dependency inversion

    public BookService( BookRepository repository){
        //dependency injecting constructor
    }

    public void setBookRepository(BookRepository repository){
        //setter based dependency injection
    }

    public List<Book> getAllBooks(){
        return repository.getAll();
    }
}
```

Banking MVC

Tuesday, March 7, 2023 11:47 AM

```
interface BankRepository{  
    void addAccount(BankAccount account);  
    List<BankAccount> getAllAccounts();  
    BankAccount getAccountById();  
    void updateAccount(BankAccount account);  
    void deleteAccount(int accountNumber);  
}
```

```
interface BankService{  
    void deposit(int accountNumber, double amount);  
    void withdraw(int accountNumber, double amount, String password);  
    void transfer(int source, double amount, String password, int target);  
}
```

```
class DefaultBankService implements BankService{  
  
    //do dependency injection of any repository  
    BankRepository repository;  
  
    public void deposit(int accountNumber, double amount){  
        var account = repository.getAccountById(accountNumber);  
        account.deposit(amount);  
        repository.updateAccount(account);  
    }  
}
```

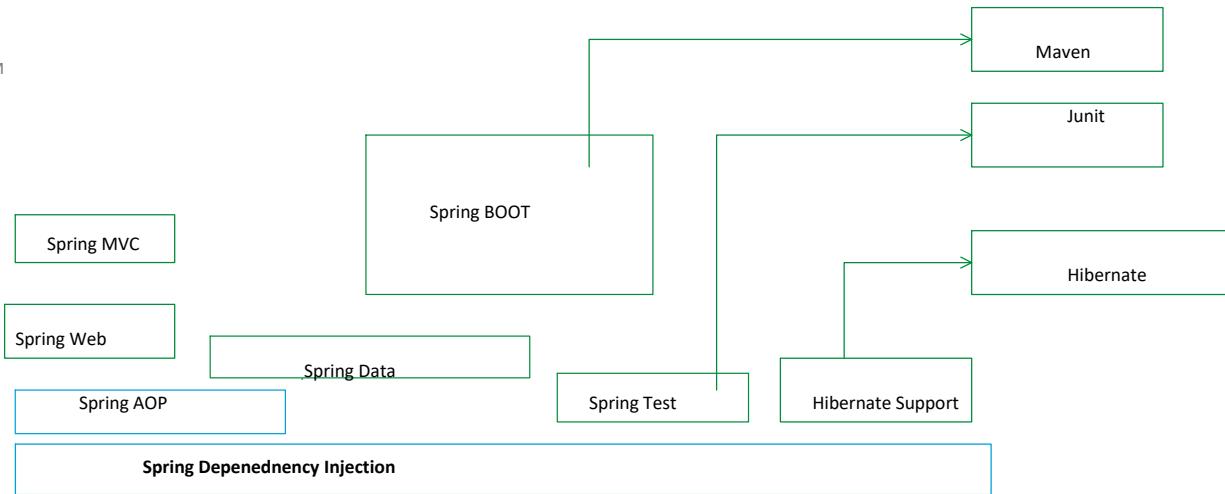
```
@Controller  
class BankAccountController{  
  
    BankService service = new DefaultBankService();  
    //inject service object here  
  
    @Post("/deposit/:id")  
    public TransactionReport deposit(int id, double amount){  
  
        service.deposit(id,amount);  
        return new SuccessTransactionReport(...);  
    }  
}
```

```
class InMemoryBankRepository implements BankRepository{  
  
    HashMap<Integer, BankAccount> accounts;  
  
    //store only in the hashmap  
}  
  
class JdbcBankRepository implements BankRepository{  
  
    DbManager manager;  
}
```

- Here controller needs a service
 - someone should add the service object here
 - It shouldn't be done using constructor call
 - why?
 - every request may create a new service (life cycle management)
 - we need to know the dependency of service on repository
 - tomorrow we may change service object or repository object
 - ◆ then we need to change all related controllers

Spring

Tuesday, March 7, 2023 12:00 PM



Hibernate

Tuesday, March 7, 2023 2:58 PM

ORM (Object Relation Mapping)

- Allows us to map our Objects to database
 - we can access database as a collection of objects rather than as table and sql
 - `select * from books`
 - get a list of Book

Object Oriented Programming

```
class Author{  
    String id;  
    String name;  
    String biography;  
    String email;  
    int age;  
    String photo;  
    List<Book> books;  
}  
  
class Book{
```

```
    String id;  
    String title;  
    Author author;  
    int price;  
    String cover;  
}
```

RDBMS

```
create Table AUTHORS(  
    name VARCHAR(200),  
    biography VARCHAR(2000),  
    email VARCHAR(200),  
    age INT,  
    photo varchar(300)  
    ID VARCHAR(100) PRIMARY KEY NOT NULL  
    //No Equilvalent of books here  
    //We may need a reverse query  
)
```

```
create Table Book(  
    id VARCHAR(200) PRIMARY KEY NOT NULL,  
    TITLE VARCHAR(200),  
    AUTHOR_ID VARCHAR(200) FOREIGN  
    KEY(AUTHOR.ID),  
    PRICE INT,  
    COVER VARCHAR(200);  
)
```

Traditional Approach (JDBC)

- Java makes a connection to database
- To Get Records
 - WE fire select query to get ResultSet
 - We need to manually covert result set to Java Object
- To Update records
 - We manually need to convert Java Object to insert/update query
 - `executeUpdate` with the query
- Problem
 - manual conversion between object and query
 - each database handles query differently
 - handling pre-fetched records may be difficult
 - can we cache the records we already got
 - How to optimize update

Hibernate

- introduced a way to auto map between database and object
- We need to define

- which object will be persisted in which table
 - which column will map to which field of the object
 - This mapping was originally done using Xml configuration
 - Now mappings are done using annotations and convention.
- Once defined we can
 - request for objects
 - update /save objects
 - we may not need to fire sql query
 - Hibernate provides it's own object based query called HQL (Hibernate Query Language)
 - The mapping was

JPA (Java Persistent API)

- JPA is a specification for implementing ORM
- Defined by Oracle as a standard
- It defines a set of mapping annotation
- No class is present
- Different frameworks can implement JPA
- Hibernate follows JPA standard.