ANZ Java

Wednesday, February 1, 2023　　10:59 PM

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

http://tiny.cc/anz-java

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

C++

Thursday, February 2, 2023   8:46 AM

- C with class
- new C

C ++ --

C = C+1

X++

C = C+1

Java

Thursday, February 2, 2023   8:54 AM

Java
- **Platform independent**
- **Architectural neutral**
- General purpose
- Object Oriented
- Multi-threaded
- Network
- **secured**
- robust
- high peromance
- **interpreted**
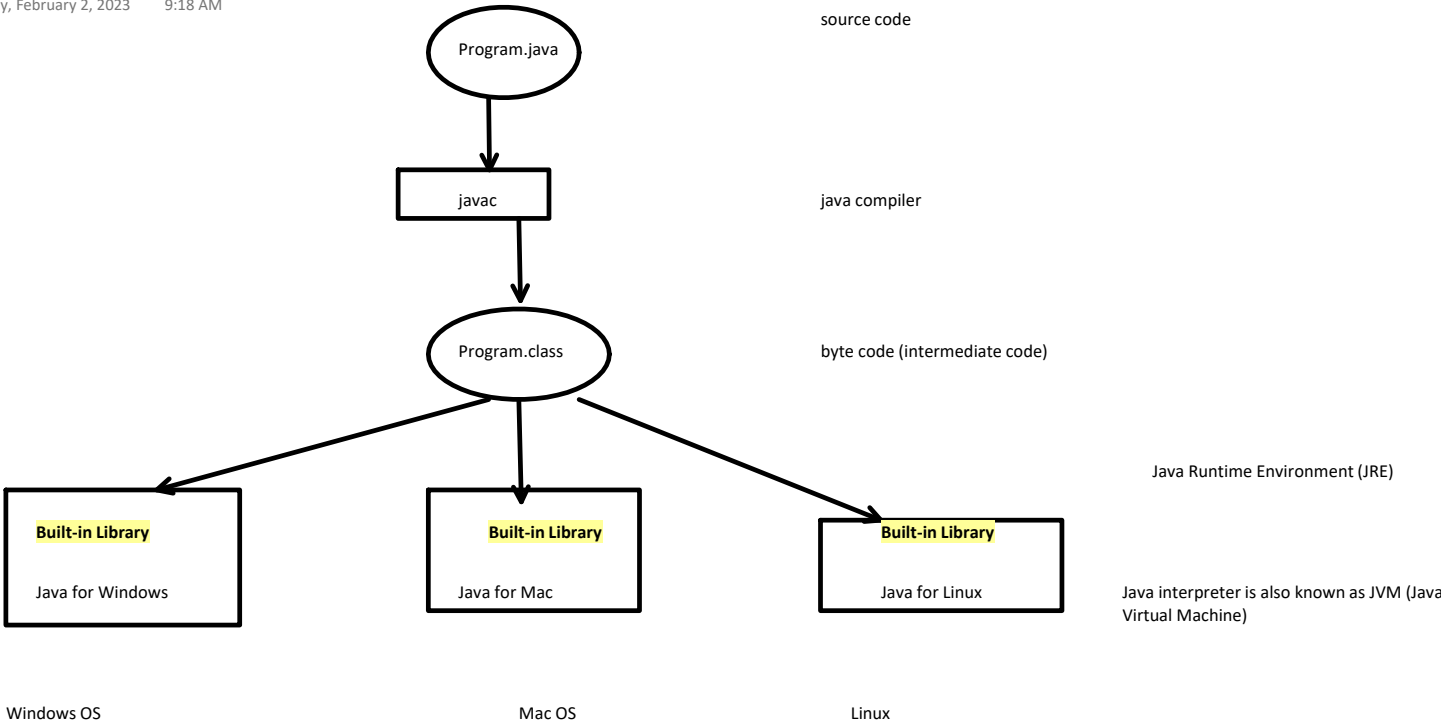
programming language.

Java Promise

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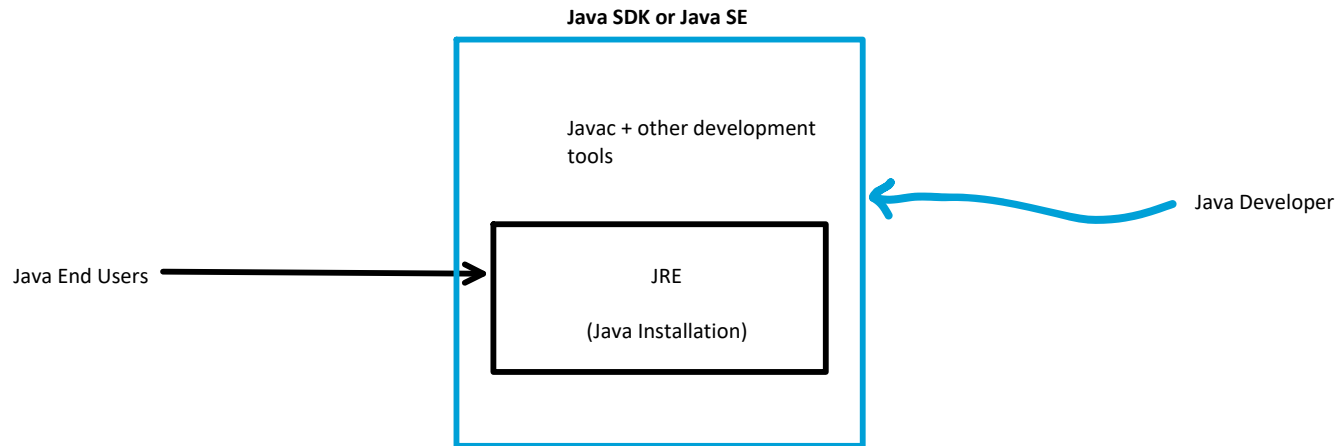- No separate code for different OS/Hardware combination.

Java Program Flow

Program.java → source code

javac → java compiler

Program.class → byte code (intermediate code)

Java Runtime Environment (JRE)

Built-in Library

Java for Windows

Built-in Library

Java for Mac

Built-in Library

Java for Linux

Java interpreter is also known as JVM (Java Virtual Machine)

Windows OS

Mac OS

Linux
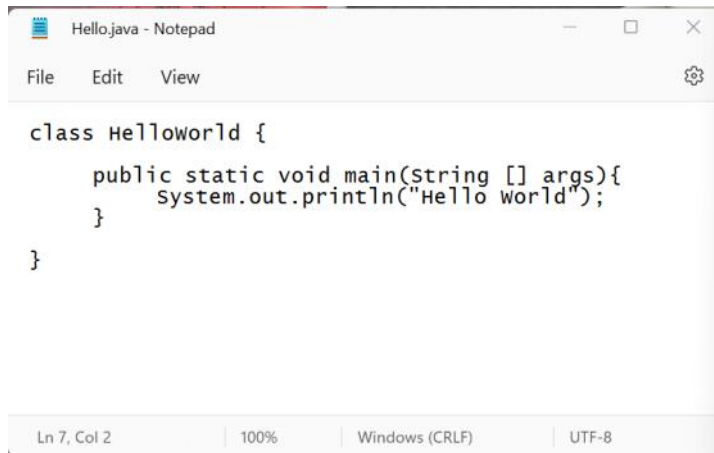
Installation and Bundles

Java SDK or Java SE

Javac + other development tools

JRE

(Java Installation)

Java End Users

Java Developer
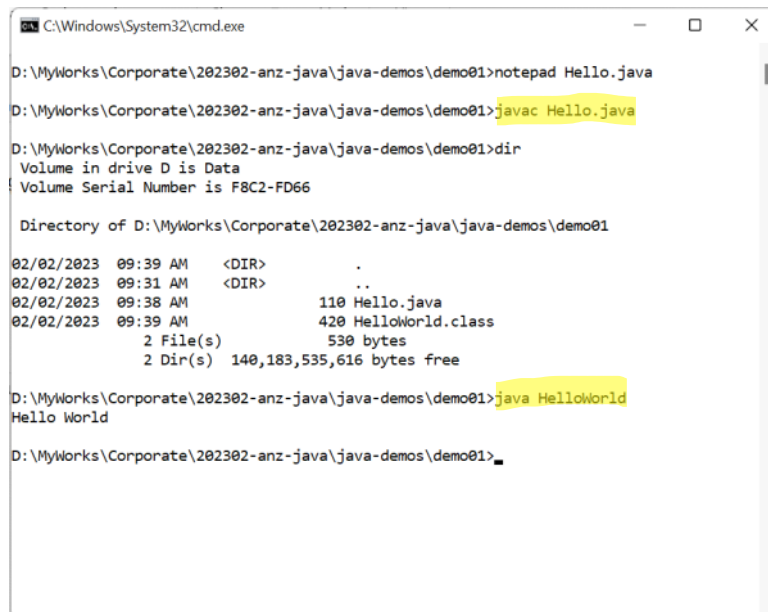
Hello World

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>_
```
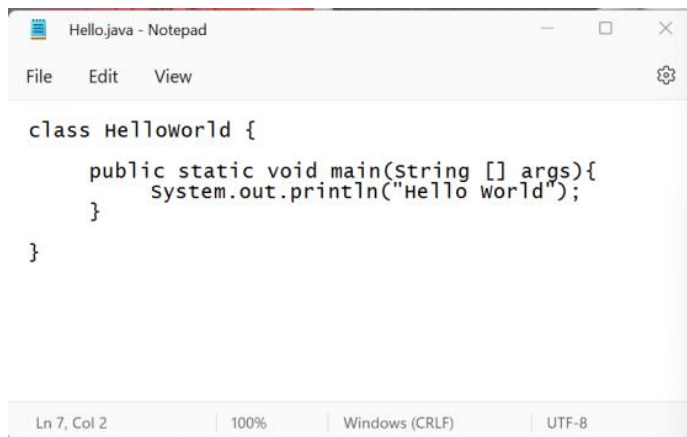
Step #1 compile

- 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 sensetive without suffixing .class

Basic Java

Thursday, February 2, 2023   10:34 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
```

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 sensetive.
  ○ 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 orginal defintion.
      □ 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= 'ॐ'; // 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 variabel
```

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

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

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 precession)
  - double
    - floating point decimal number, double precession
  - 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 arithematic quality

    - byte
      - represents a single byte
    - short
      - short int
    - long
      - long int

- **String**
  - String is a series of char to represent
    - word
    - sentense
  - It is double qutoed
  - Note String begins with upper case S
    - It is a class and not a keyword
    - It is a predefined class created by Java team

```
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);
```

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\simple-demos>javac ArithmeticApp01.java
ArithmeticApp01.java:12: error: incompatible types: boolean cannot be converted to int
        x=false;
          ^
1 error
```

Compatible and Incompatible type

- Few types are compataible 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
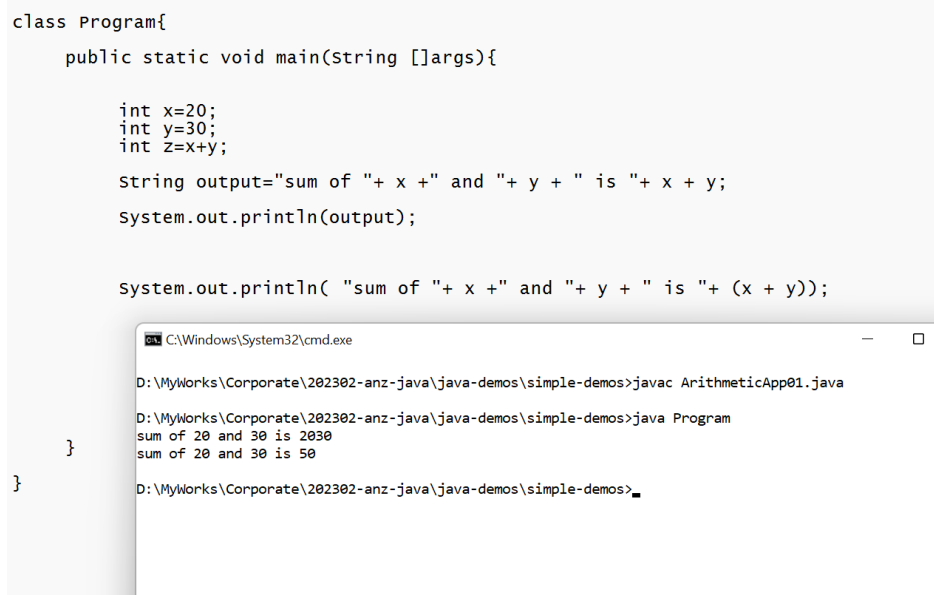 - Java doesn't allow this conversion automatically
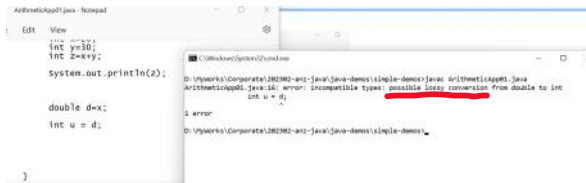
- we can force such conversion by explicit type casting

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

- wat 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);
```



```
}
}
```

```
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));
```



```
    }
}
```

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
- Whereever 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

Increment and Decrement (Prefix and Post fix)

- when increment/decrement is a independent expression they are exactly same
  - x++;
  - ++x

- when increment/decrement comes as part of another expression
  - prefix is resolved before resolving the expression
  - postfix is resolved after resolving the expression

```
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 with become 6*10 = 60

Integer Operations

```
int x=50;
int y=4;
```

- 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 => 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)

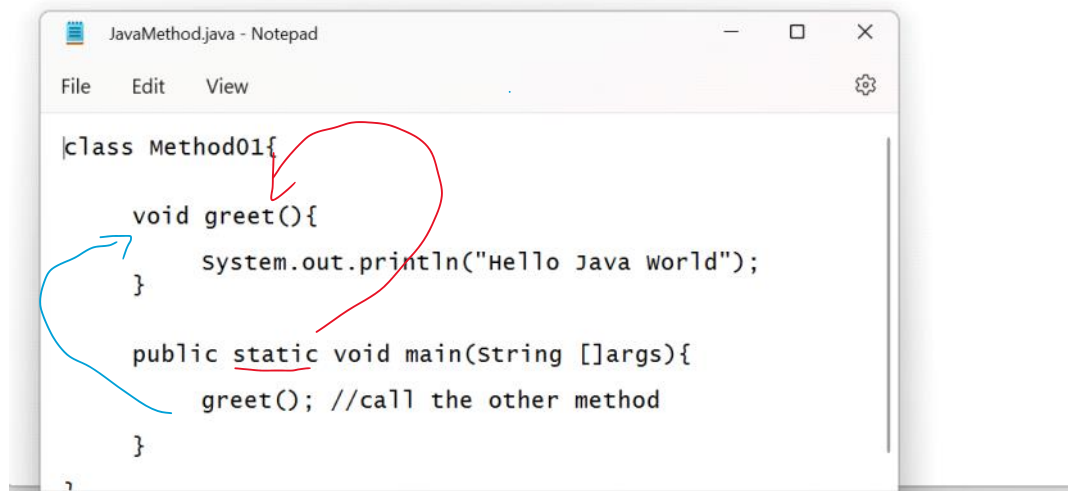Thursday, February 2, 2023   12:45 PM

- 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 exeuctable code


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

):\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

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

```
JavaMethod.java - Notepad                          —    □    X

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!

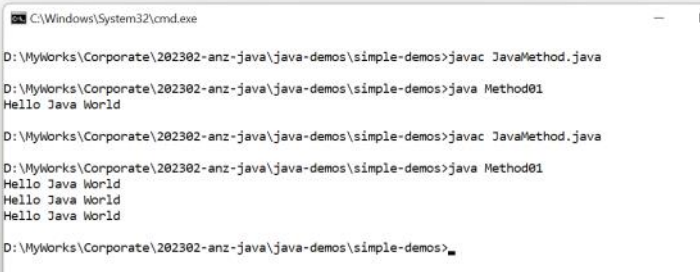## Working with multiple Methods

```
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

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

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

Note:

- We have two methods in our program
  - greet
  - main
- Eventhough "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
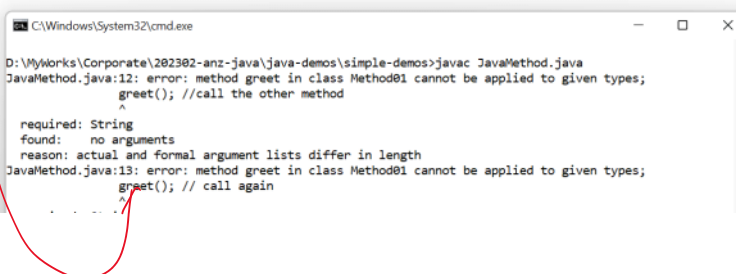 - ...

What if we need to greet someone specific?

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

```
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
                ^
```

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

```
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

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

- 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

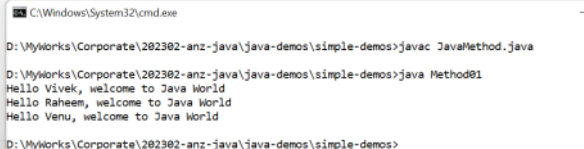```
File   Edit   View

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
    }
}
```

```
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

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
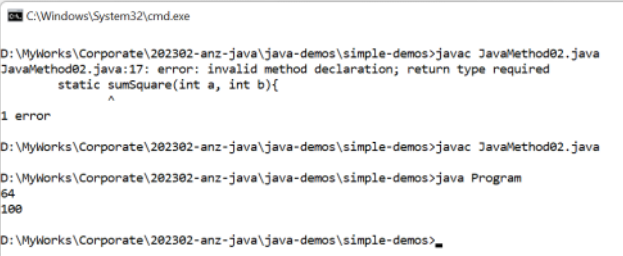
```
class Program{


    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;
    }
}
```

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

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 forumla
    - int c=sumSquare(5,5)*10;
  - print directly
    - System.out.println(sumSquare(2,3));

- each method has it's 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

```
if ( boolean_expression){

    statement1;
    statement2;
}


if(boolean_expression)
    single_statement;
```

Important

- block marker is always required in class and method even if they contain single statement
- for loops and branching braces are optional if there is 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( initalization;  condtion;  reinitialization)
    block or statement;
```

- for executes in othe order
    1. runs intialization
    2. checks condition
    3. runs block or statement if conditon is true else exists
    4. runs reinitalization
    5. repeats from step 2

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

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

- 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 ramaining 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

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;

}
```

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

- 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)

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 compulsary 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

```
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 caleld
- Use Case
    - A Dictionary class can be used as
        - stand alone dictionary app
        - embedded in Word to spell check.

Object Oriented Program

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();
```

- 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
```

- An object can also have it's own behavior or roles

```
class List{
    int items;
```

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)

Note
- class doesn't have the price variable
  - It has the definion 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 intially assigned to all chairs

  - each chair can individually change it
    a

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

```java
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

```java
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

```
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

```java
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("Furntitures: %d\n", furnitures.count());

    Chair c1=new Chair();
    Chair c2=new Chair();

    c1.price=3000;
    System.out.printf("c1.price=%d\tc2.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);
```

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 declreases 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

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

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()

```java
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);
```

- 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 Intialized 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 initalized to
      - null for objects
      - 0 for numbers
      - false for boolean
      - '' for char
  - Method local variables
    - They remain uninialized till you initalize them
    - You can't use them without first initializing them

```java
class Program{

    int number; //by default 0
    String name; //by default null;


    void f1(){
        int x; //un-initalized
        String y; //unitialized


        y="Hi";

        System.out.println(y); //works

        System.out.println(x); //fails. not
        intilized

        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 incoude 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

```
D:.
└───src
        FurnitureApp.java

    ├───data
    │       List.java

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

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


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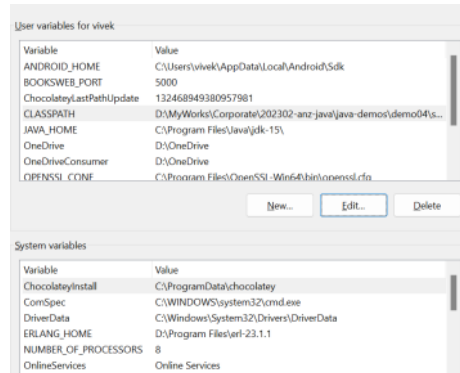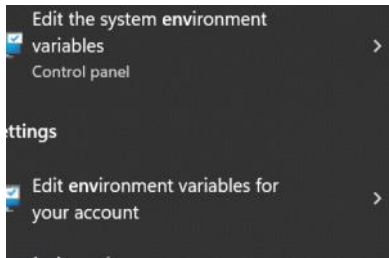
```
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src>echo %classpath%
D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src\furnitures;D:\MyWorks\Corporate\202302-anz-java\java-demos\de
mo04\src\data;D:\MyWorks\Corporate\202302-anz-java\java-demos\demo04\src\finance

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

```
Volume serial number is F8C2-FD66
D:.
└───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
```

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

- By default Java/javac searches for class (both source/bytecode) 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

No classpath set

```
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 Furnitur
eApp
Total Customers: 1
Furntitures: 3
c1.price=3000    c2.price=2000
Table@776ec8df
Bed@4eec7777
Inventory@3b07d329
Invoice@41629346
Invoice@41629346
Invoice@404b9385
customerList [  Vivek   ]
furntiures [    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>
```

CLASSPATH is set using -cp switch

How to separate source and class files in different folders

Class name conflict

What is a Table?

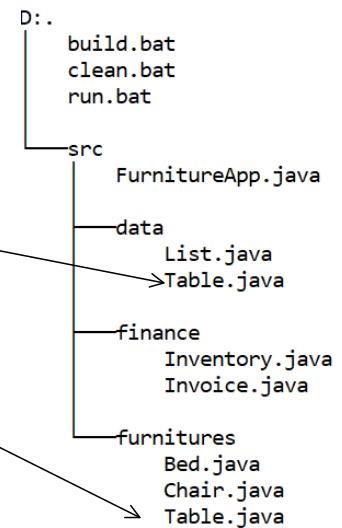
Or

Data Table   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);
```

```
D:.
    build.bat
    clean.bat
    run.bat

    src
        FurnitureApp.java

    data
        List.java
        Table.java

    finance
        Inventory.java
        Invoice.java

    furnitures
        Bed.java
        Chair.java
        Table.java
```

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
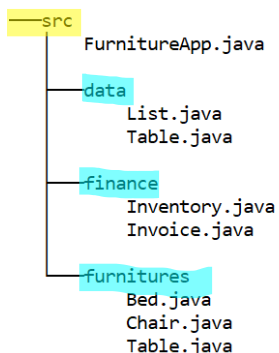
Friday, February 3, 2023   12:39 PM

- 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 packages lives inside a folder

How do we create a package

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

```
—src
    FurnitureApp.java
    —data
        List.java
        Table.java
    —finance
        Inventory.java
        Invoice.java
    —furnitures
        Bed.java
        Chair.java
        Table.java
```

- Here we have designated following folder as packages
  - data
  - finance
  - furnitures
- A package becomes language concpet 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
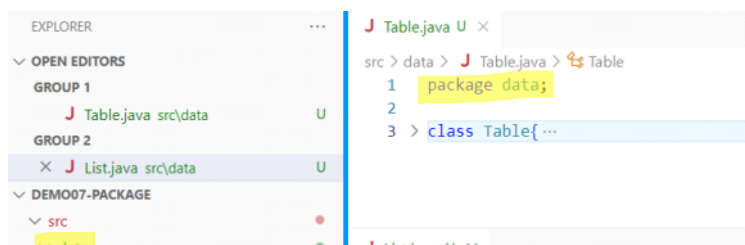
- package "data" should match the immeidate folder structure
  - more important for .class file than .java file
- if we chose to name our package as src.data
  - src package will contain subpackage data
- In our example data is package, src is container path for the package.
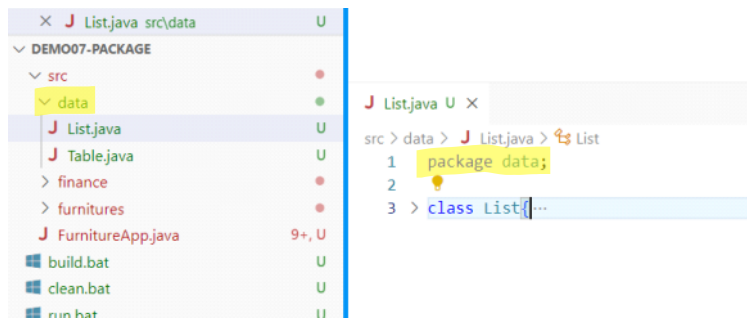
- if we chose to name our package as src.data
  - src package will contain subpackage data
- In our example data is package, src is container path for the package.

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/f
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>
```
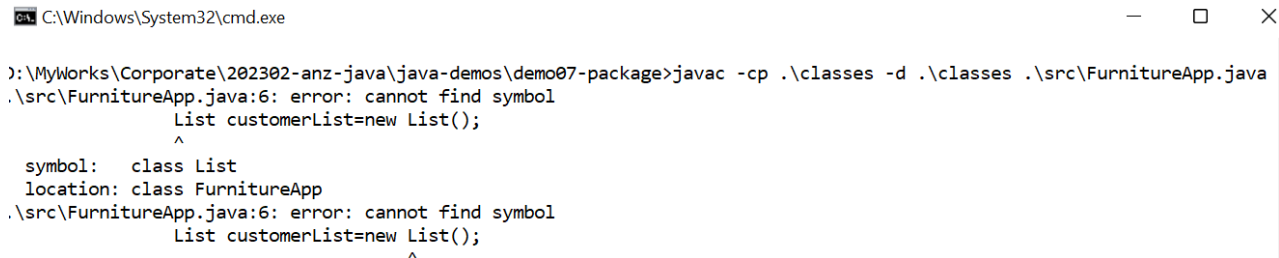
Problem

- 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

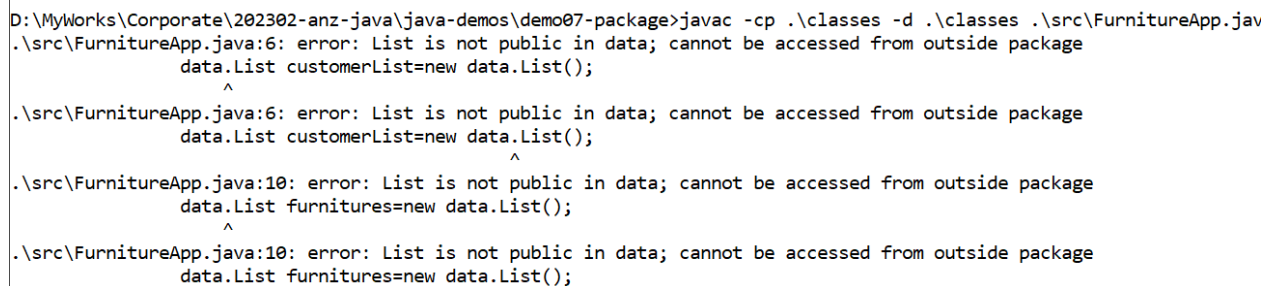## Problem 2

```
C:\Windows\System32\cmd.exe                                    —    □    ×

):\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();
                                      ^
```
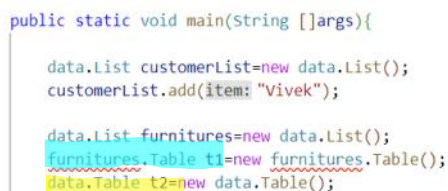
- Now It is searching for List class in classes folder
  - It doesn't exist
- In fact we don't have a global List class
  - we have data.List

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.jav
.\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 unamed 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 propely
- 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 pakcage contents (classes) directly so that we can use them without qualfiying the packae name

```java
import finance.*; //get all the classes from finance packae

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 pacakge 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 packae
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 prorammer 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

- folder
- package

- **src**
  - ○ class App
  - ○ **vivek**
    - ▪ **data**
      - □ class List

- □ class Tree
  - ▪ **furnitures**
    - □ class Chair
- ○ **sanjay**
  - ▪ **data**
    - □ class Search
    - □ class Table
    - □ class List

How do I access both Tree and Search class?

- we can have both src\sanjay 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\sanjay;.\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 boundry
  - ○ 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 possiblity 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

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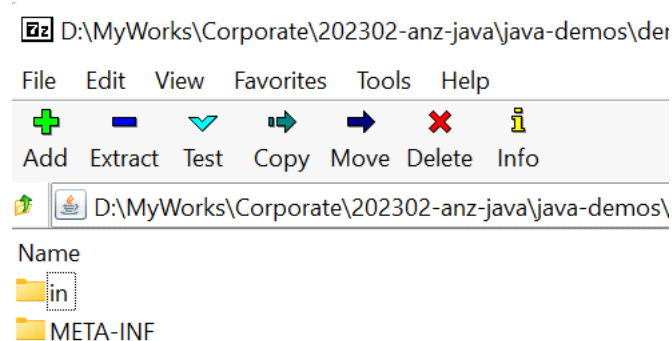
- A jar contains all my files
- It also include some Meta information needed by Java

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
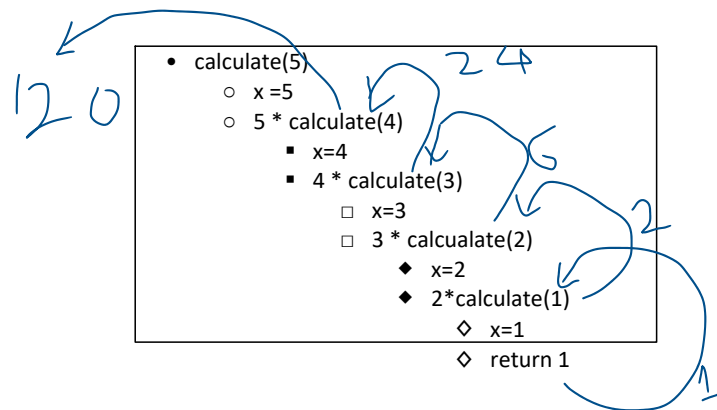- Example

- Factorial of 5

5 x **4 x 3 x 2 x 1**

Factorial of 4

- Now we can say

5! = 5 * 4!

- Programmetically we can write

```java
package in.concpetarchitect.maths;

public class Factorial {

    int calculate(int x) {
        if(x<=1)
            return 1;
        else
            return x* calculate(x-1);
    }

}
```

- calculate(5)
  - x =5
  - 5 * calculate(4)
    - x=4
    - 4 * calculate(3)
      - x=3
      - 3 * calcualate(2)
        - x=2
        - 2*calculate(1)
          - x=1
          - return 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
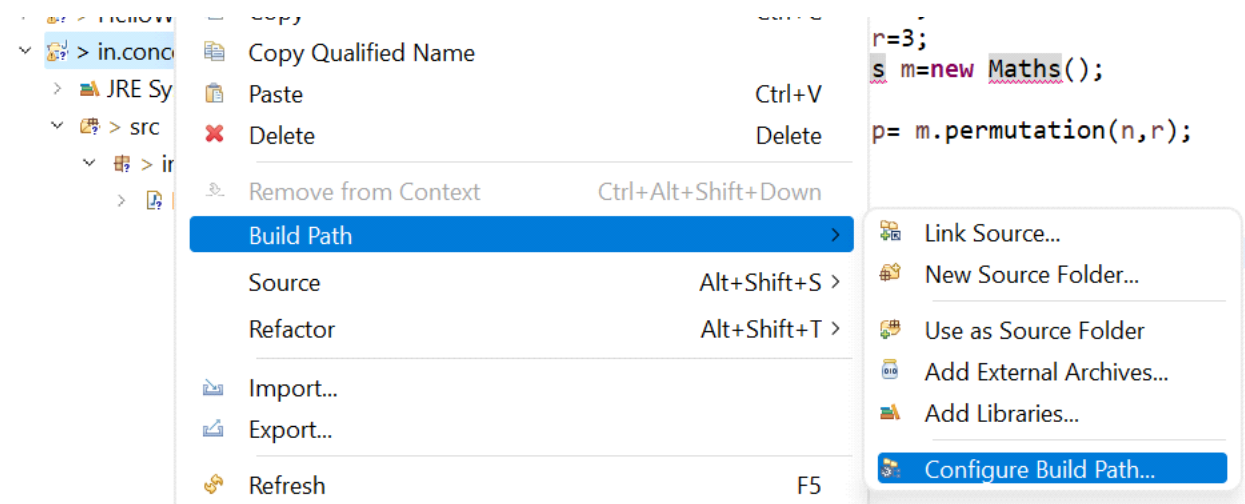
Monday, February 6, 2023   10:24 AM

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
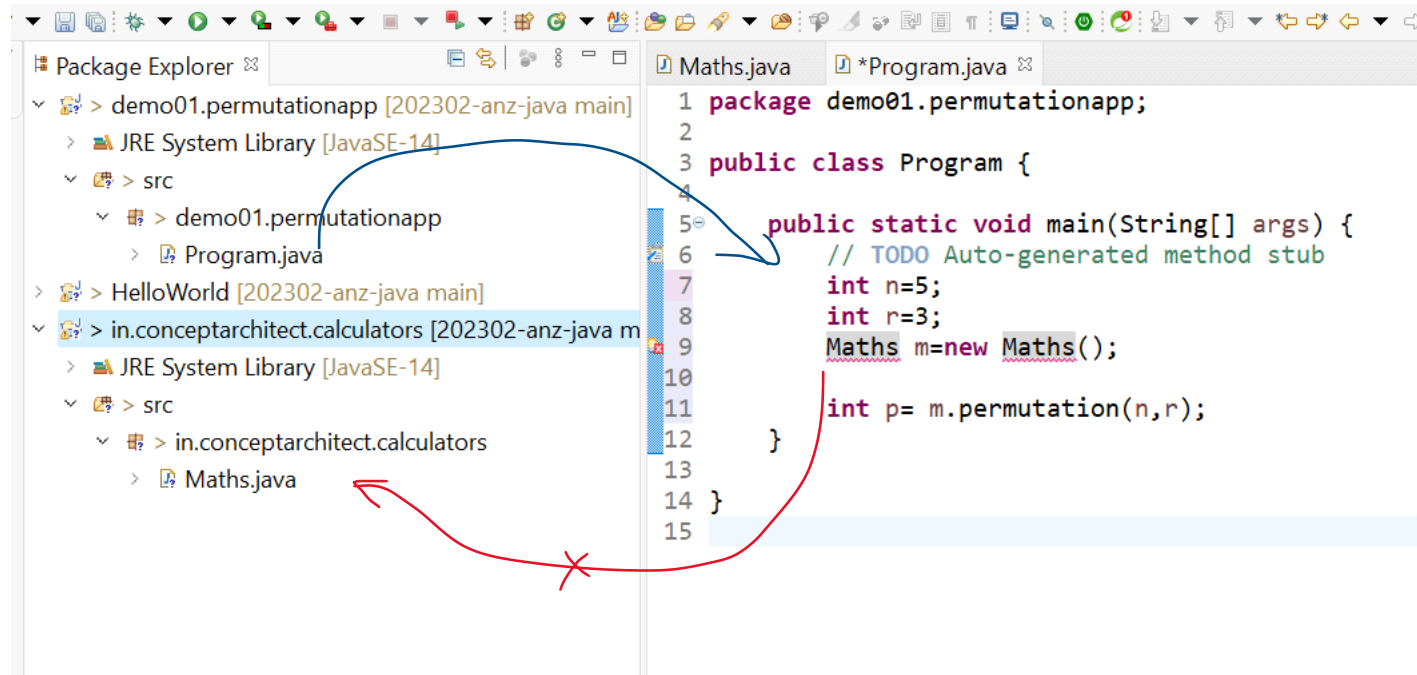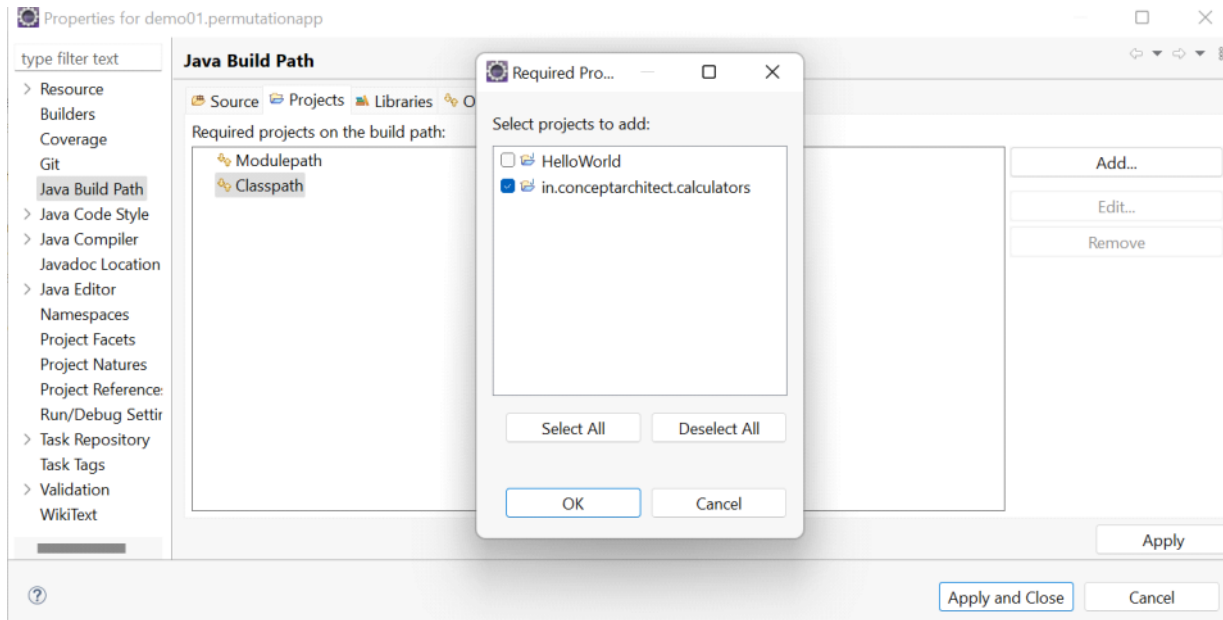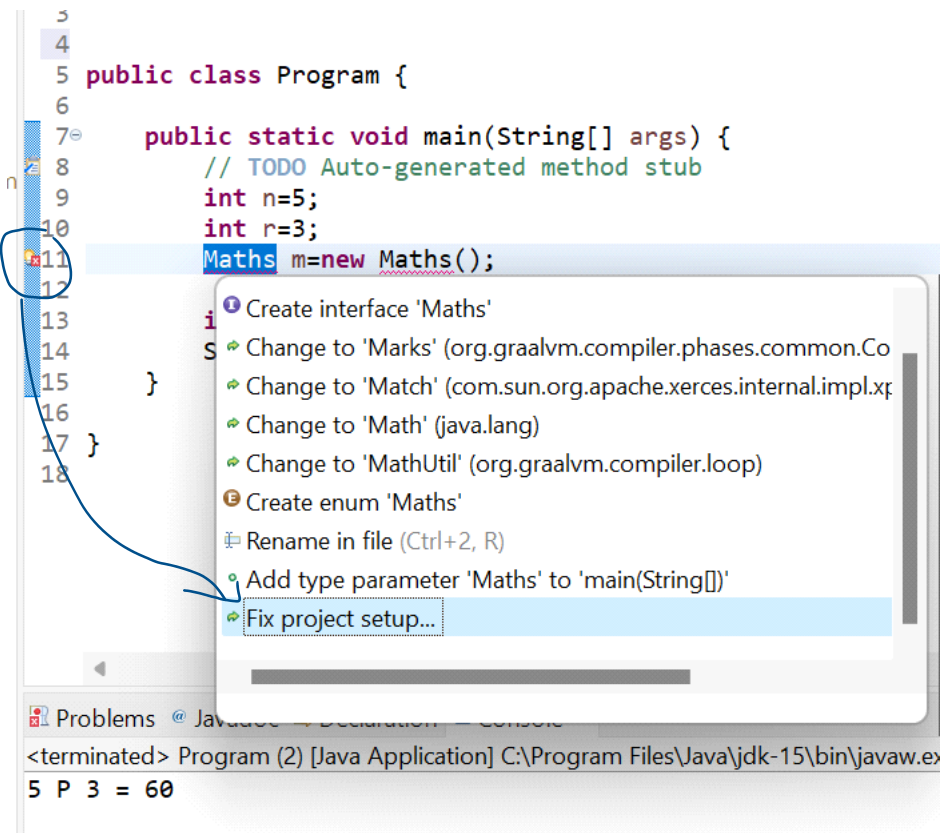

Adding Reference of One Project in another
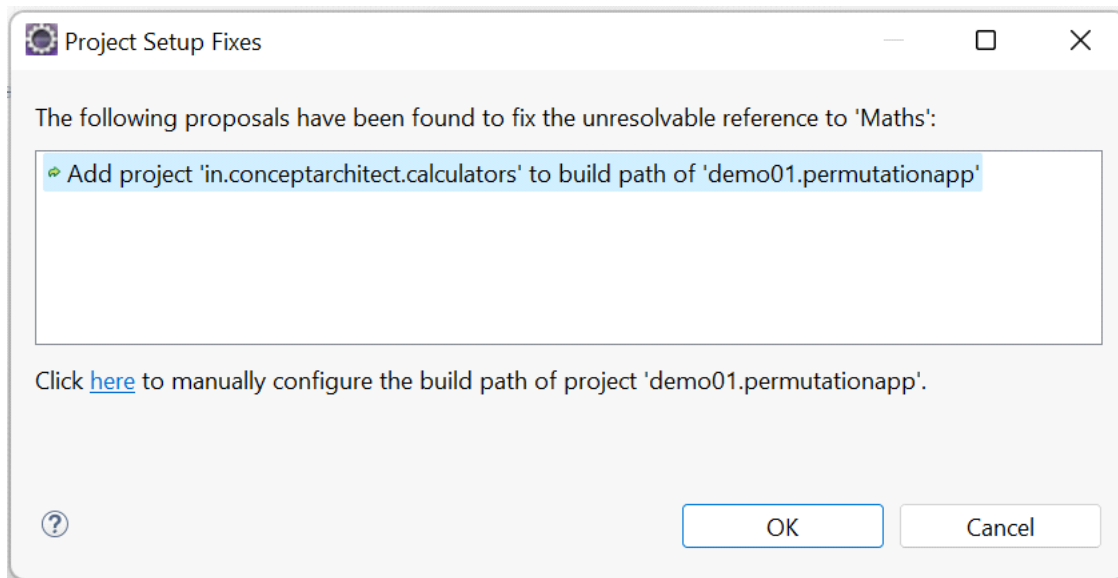
1. Right click project and select the option

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'.
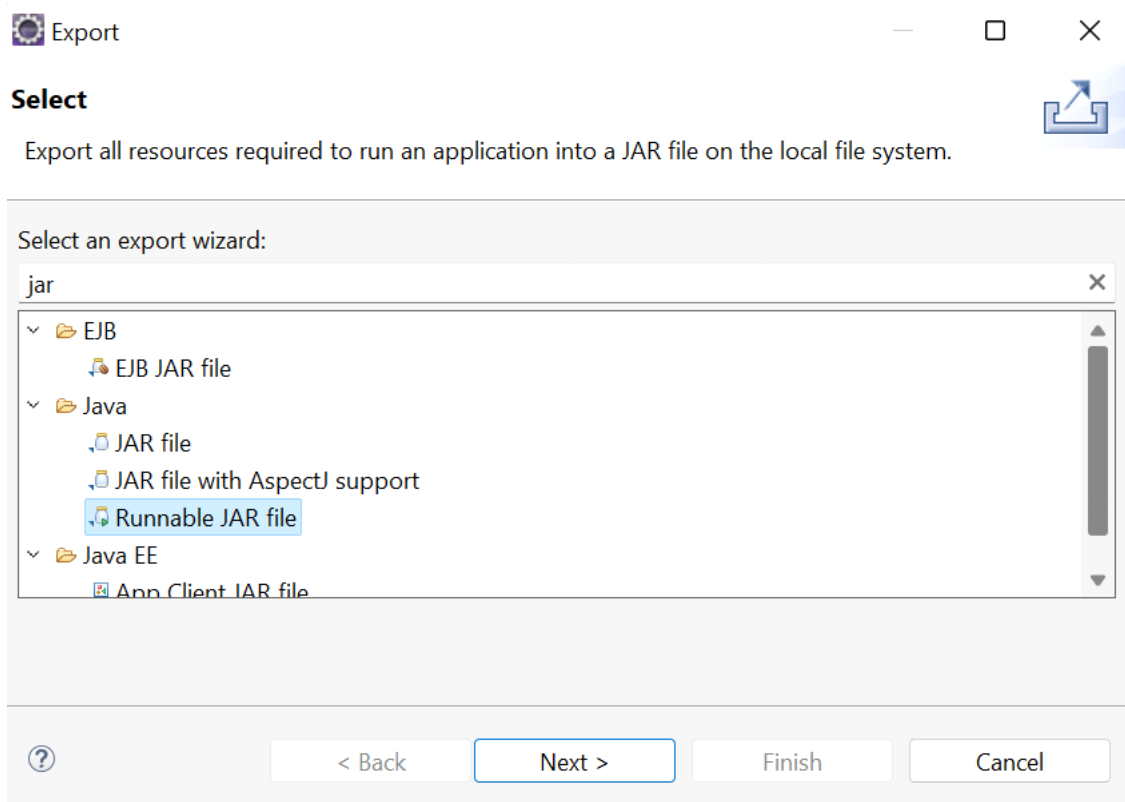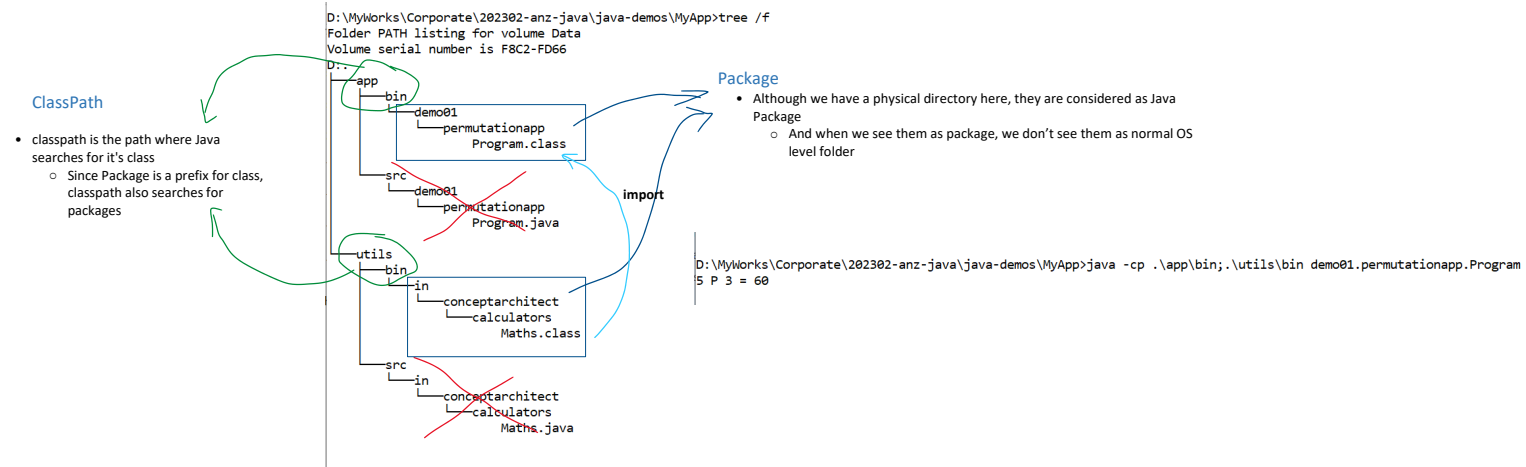
OK   Cancel

Creating jar from eclipse

Monday, February 6, 2023   10:36 AM

1. File —> export

Understanding Runtime Classpath

Monday, February 6, 2023   10:55 AM

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\MyApp>tree /f
Folder PATH listing for volume Data
Volume serial number is F8C2-FD66
D:.
├───app
│   ├───bin
│   │   └───demo01
│   │       └───permutationapp
│   │               Program.class
│   │
│   └───src
│       └───demo01
│           └───permutationapp
│                   Program.java
│
└───utils
    ├───bin
    │   └───in
    │       └───conceptarchitect
    │           └───calculators
    │                   Maths.class
    │
    └───src
        └───in
            └───conceptarchitect
                └───calculators
                        Maths.java
```

ClassPath

- classpath is the path where Java searches for it's class
  - Since Package is a prefix for class, classpath also searches for packages

Package

- Although we have a physical directory here, they are considered as Java Package
  - And when we see them as package, we don't see them as normal OS level folder

import

```
D:\MyWorks\Corporate\202302-anz-java\java-demos\MyApp>java -cp .\app\bin;.\utils\bin demo01.permutationapp.Program
5 P 3 = 60
```

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 20

x = 30;

```
int y=20; //memory allocated and initalzed to gether
```
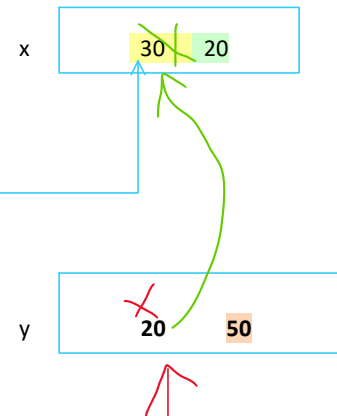
y | 20 50

Assigning another value to a variable

```
x=y;
```

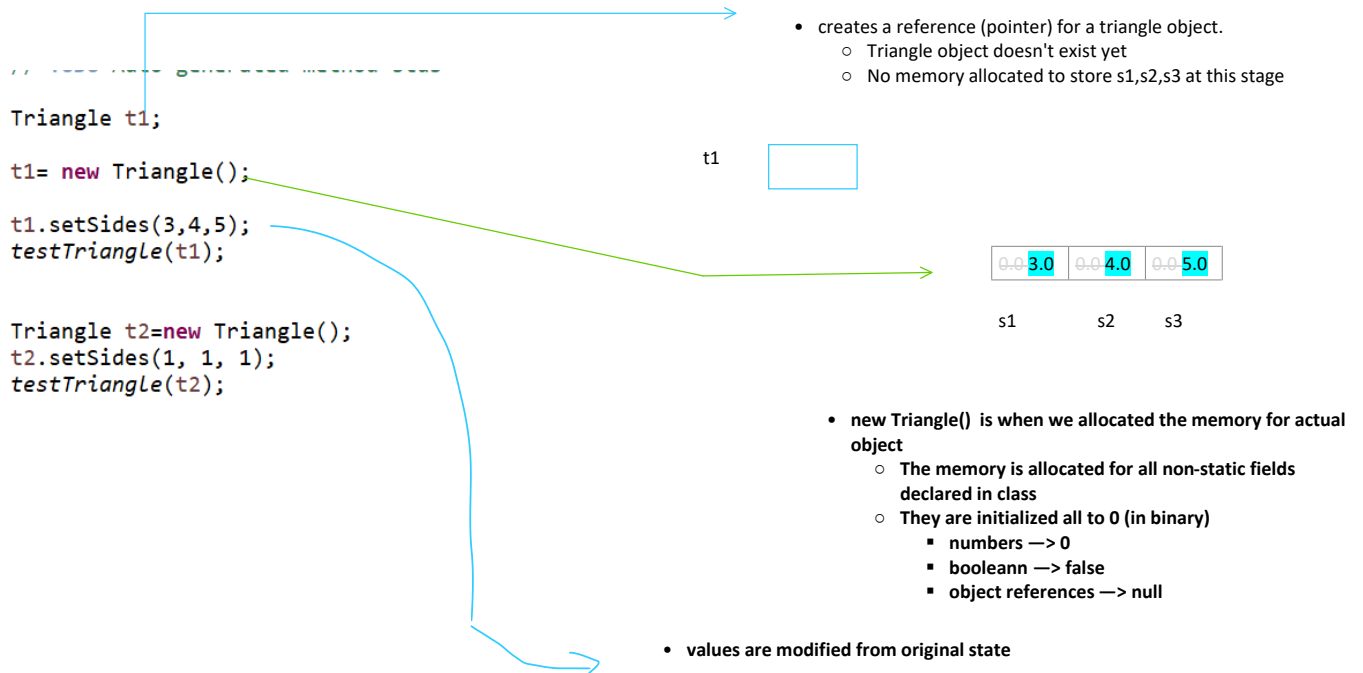- current value of y is copied to X. But they are not related any further

```
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
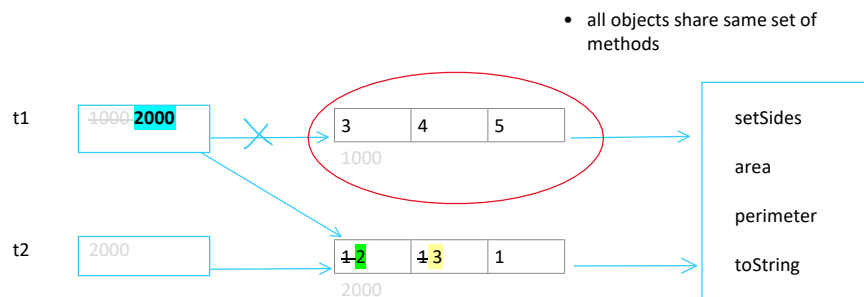
Monday, February 6, 2023   11:35 AM

```
// TODO Auto-generated method stub

Triangle t1;

t1= new Triangle();

t1.setSides(3,4,5);
testTriangle(t1);


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

- creates a reference (pointer) for a triangle object.
  - Triangle object doesn't exist yet
  - No memory allocated to store s1,s2,s3 at this stage

t1

| 0.0 3.0 | 0.0 4.0 | 0.0 5.0 |

s1   s2   s3

- **new Triangle() is when we allocated the memory for actual object**
  - **The memory is allocated for all non-static fields declared in class**
  - **They are initialized all to 0 (in binary)**
    - **numbers —> 0**
    - **booleann —> false**
    - **object references —> null**

- **values are modified from original state**

IMPORTANT NOTE

- Memory is only allocated for non-static fields declared inside the calss
- 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

- all objects share same set of methods

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



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

t1 | 1000 2000 |

| 3 | 4 | 5 |
1000

t2 | 2000 |

| 1 2 | 1 3 | 1 |
2000

| setSides |
| area |
| perimeter |
| toString |

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)
  - They 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 it's 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 gurantee IMMEDIATE
        - It is meant to be suggestive not authoratative
        - although gernerally it is IMMEDIATE
    - In MOST cases it is not recommened to interfere in the process.

Memory Layout of a typically Application
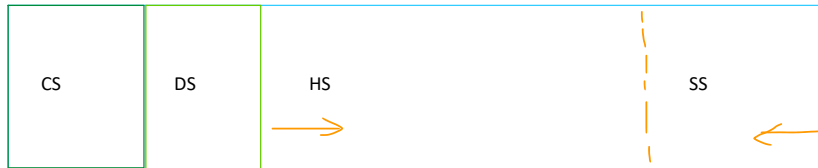
- Code Segment
  - stores your logic
    - class methods
- Data Segment
  - Stores static and const values
- Heap Segment
  - Stores dynamically allocated memory
  - "new"
- Stack Segment
  - Stores method locals and parameter
- There is no hard division between Stack and Heap
  - They grow towards each 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 object 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
  - other 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 object to gen 1
      - Note address will change and that is why java never gives the address to program
      - All the references to this object is 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

```
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

```java
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     }
```

- 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

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

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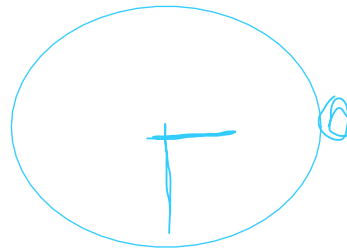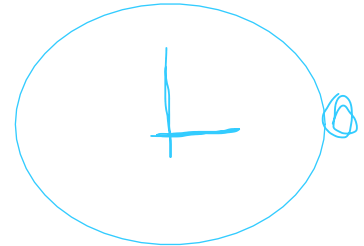
- we may bypass setSides and change values directly
- Now s1 has changed but valid flag is not reset

```
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);
```

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

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

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 disucssed 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 inval
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 temper with this information
- This information will be set only using right set of values

Problem —> what if we need to see the value

```java
private static void testTriangle(Triangle triangle) {
    System.out.println(triangle);
    if(triangle.valid) {
        System.out.print("Perimeter: "+triangle.perimete
        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.

```java
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?

```java
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?

```java
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 exists with side 0?

- If Triangle is created on Line 14
  - What was the value of "t1" on line 13?
  - what is we can 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
  - There creations are no 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

Assignment 3.1

- 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
            - balance = balance*rate/1200
    - to string
        - to show the account object as string

- Write a test app to work with bank account

'this'

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

Triangle t2= new Triangle(6,8,7);
```

t1			3	4	5

t2			6	8	7

area()

set(double x,double y, double z)

perimeter()

```
var x= t1.s1;

var y= t2.s1;

var p1= t1.perimeter();

var p2= t2.perimeter();
```

- both object calls the very same method that apparently doesn't take any parameter
- How will the method know which object we are talking about?

'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)

```
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();
    }

}
```

```
//conceptual view

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

```java
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
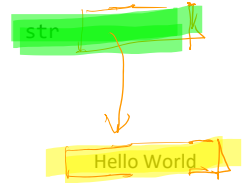 cash"
    ○ we want the cash

String class

Tuesday, February 7, 2023   11:18 AM

- String is not a premitive 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

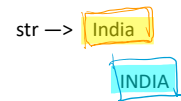`String str;`

`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`
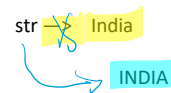
str —> India
        INDIA

- 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();`
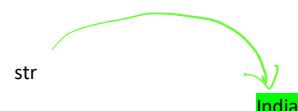
str —✗ India
        INDIA

- 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 importan!

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

str

India

`String str="India";`

```
String str="India";

String str2="India";
```

str

India

str2

- 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 rerefernce 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.
    - == wll return false
    - .equals will return true

- We prefer .equals over == for String comparision
  -

- 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

- compares compares two String to find how different they are from each other
- It returns the difference for first mis-match 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 interpetation
  - 0 —> equals
  - negative —> first string comes first in dictionary/alphabatic sequence
  - positive —> first string comes later in the sequence

Case Sensetive tests

- Normally Strings are case sensetive
  - "India".equals("INDIA") –> false
  - "India".compareTo("INDIA") --> 'n' -'N' = 32

- We have case insensetive comparsions
  - "India".equalsIgnoreCase("INDIA") —> true
  - "India".comparesToIgnoreCase("INDIA") —> 0

Common String class methods

Method Name	Purpose	Example

length()	returns length of string	
equals	true/false	
equalsInoreCase		
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 lenfth	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+="*";
```

str

```
""
```

```
"*"
```

```
"**"
```

```
"***"
```

Ready for garbage collection.
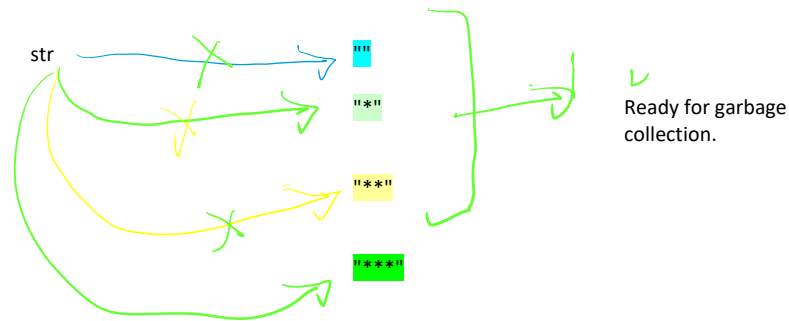
Problem

- If we have a large String maniuplation, 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 maniuplation consider to use the class StringBuilder

StringBuilder

- StringBuilder is a class that has methods to maniuplate 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

- 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 premitive 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 spcifiy 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.

arr1   arr2

0	0	0	0	0
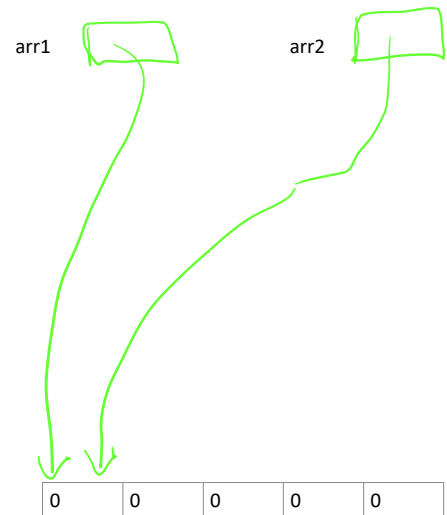
Creating Array reference and Object together

Option#1

```
int [] list1 = new int [5] ; //array of 5 items all are 0

int  list2 []  = {2,3,5,9,2};  //aray of 5 items with specified values
```

list1

0	0	3	0	0

list2

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 is what you expected.
- we must manual maintain a track of what I expect

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

- It is difficult to draw a boundry 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

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

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 poluting the test data with unexpected value

```java
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("Withdrw shoudl fail for insufficent balance",a1, amount+1, password, BankingStatus.insufficientBalance);
    withdrawTests("Withdraw should pass for happy case",a1,1,password,BankingStatus.success);

    `
```

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

```
FAILED: Withdrw shoudl fail for insufficent balance
        expected: insufficientBalance found: success
```

Unit Testing Architecture

Tuesday, February 7, 2023   1:47 PM

```
public static void main(){
    test1();
    test2();
    test3();
}
```

System.out.println()

<<Code to be tested>>

withdraw()

2. test method calls method being tested

- You need if and System.out.println
- Same program will run multiple tests
  - you may need to comment un comment tests
  - tests may interfere with each other causing
    - false negative
    - false positive

```
void test1(){

}

void test2(){   .

}
```

1. Test Runner runs all our test methods
   - each method runs in an isolated environment
   -

- No main function
  - we may have multiple independent test method
  - each test method may test a particular aspect
- No prints or if

3. test method calls assert. when assert fails test fails

TestRunner

Assert Methods

- replacement of traditional print and if
- Asserts against the values
- Automatically generates test results

4. generates and displays report of all tests executed.

Bottom Line

- A test is still a Java method
- You can write anything here what you can write anywhere
- We generally avoid direct print or if
  - we replace with assert
 - we may still use print (not recommeneded)

TestReporter

- Forms a typical Test Enviornment
- There can additional helper library to make tests better and more efficient
- Any of these components canbe 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

- 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.
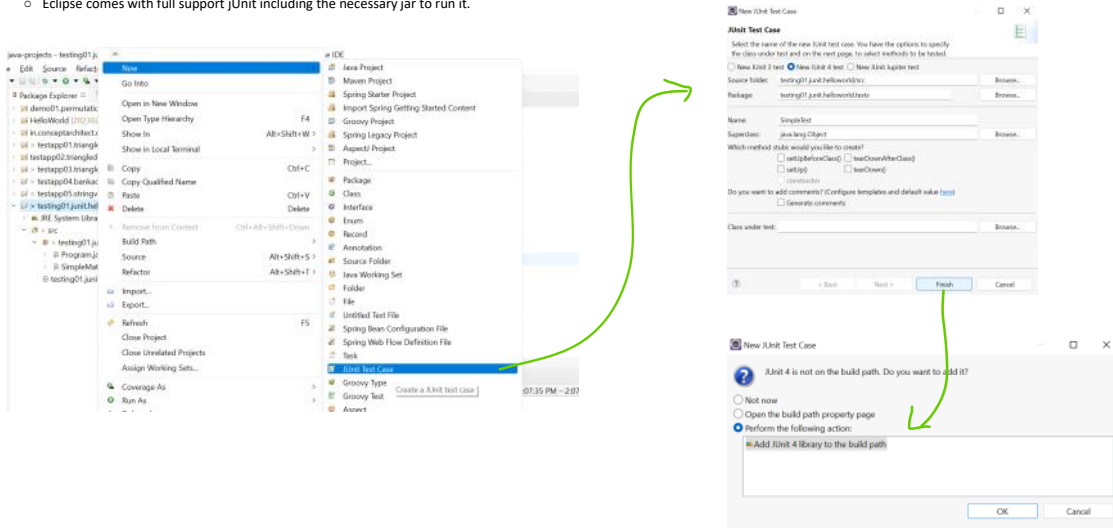
Identifying a Test

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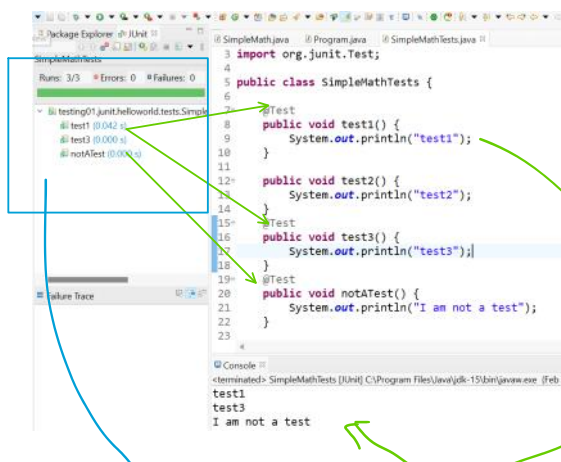- Eclipse recognizes a class with one or more test method and runs it as a test
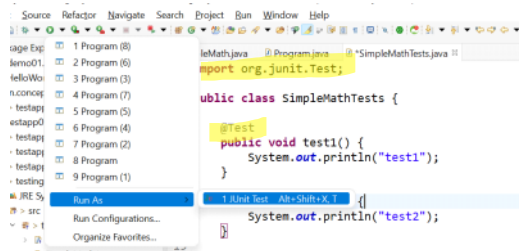
- Any method marked with @Test is a a test
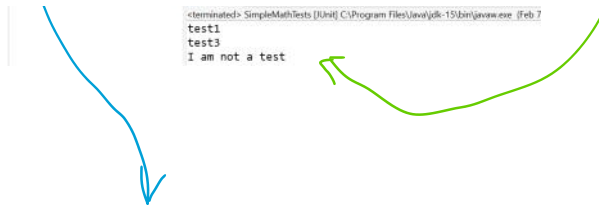
What is an annotation?

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

Running first set of Tests

Note

1. every method marked as @Test is executed
   a. name doesn't matter

2. each of these methods run independendly and in no particular order
   a. we may see their output in console

```
<terminated> SimpleMathTests [JUnit] C:\Program Files\Java\jdk-15\bin\javaw.exe  (Feb 7
test1
test3
I am not a test
```

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

```java
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);
    }
```
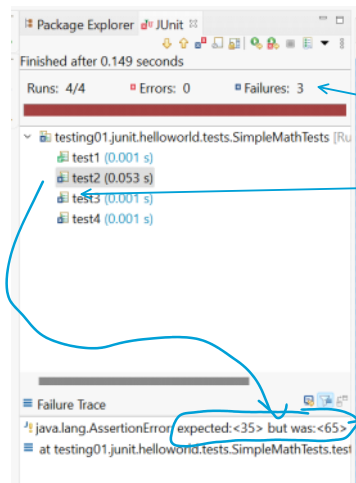
Aside Static Import

- allows us to import a static method from a class

```java
//importing a static method from class
//now we can use this static method without using class reference
import static org.junit.Assert.assertEquals;
```

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

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

```
Package Explorer   JUnit
Finished after 0.149 seconds

Runs: 4/4        Errors: 0        Failures: 3

  testing01.junit.helloworld.tests.SimpleMathTests [Ru
    test1 (0.001 s)
    test2 (0.053 s)
    test3 (0.001 s)
    test4 (0.001 s)

Failure Trace
java.lang.AssertionError expected:<35> but was:<65>
at testing01.junit.helloworld.tests.SimpleMathTests.test
```
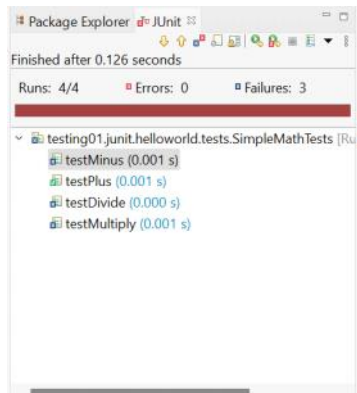
Note

- 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

```java
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

- nameing methods with sufix 1,2 may not be clear enough
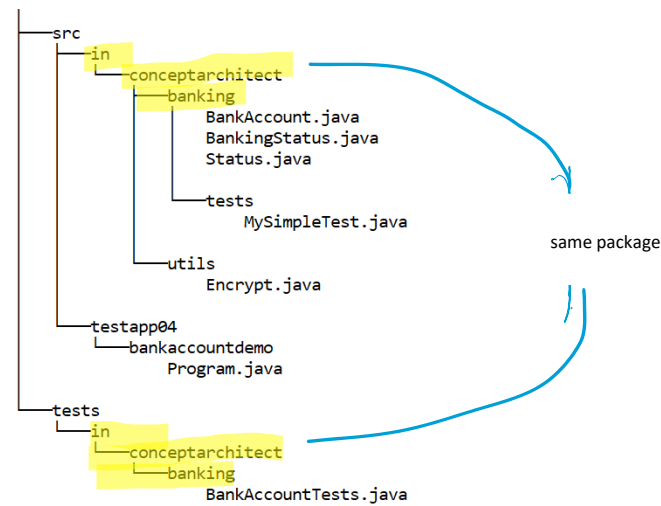 - testDeposit1
  - testDeposit2

DAMP principle (Descriptive and Meaningful Phrases)

- Method name shouldn't just be a descripted 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.conceptarchtiect.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

```
    src
    └─in
        └─conceptarchitect
            └─banking
                    BankAccount.java
                    BankingStatus.java
                    Status.java

                └─tests
                        MySimpleTest.java

            └─utils
                    Encrypt.java

    testapp04
    └─bankaccountdemo
            Program.java

    tests
    └─in
        └─conceptarchitect
            └─banking
                    BankAccountTests.java
```

same package

BankAccount memory model

```
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

- account number should be unique
- name, password, balance may or many not be similar
- **what about interest rate?**

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 fo 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);
```

b1

1	Vivek	20000	p@ss	12

b2

1	Sanjay	20000	p@ss	13

12
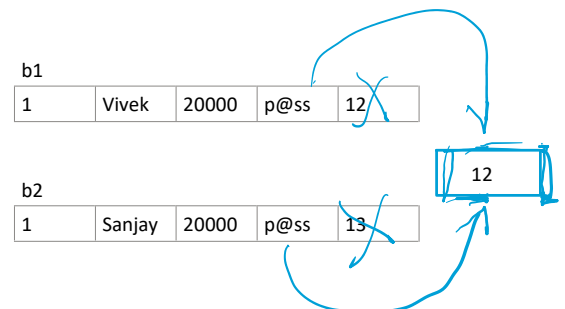
Assignment 4.1

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

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

Composite Output

```java
    public Outcome getBalance(String pass) {
        var o = new Outcome();

        if (!checkPassword(pass)) {
            o.setDescription(BankingStatus.invalidCredentials.toString());
            o.setResult(false);
        } else {
            o.setResult(true);
            o.setDoubleValue(this.balance);
        }
        return o;

    }
```

```java
 1  package com.anz.midanz.util;
 2
 3  public class Outcome {
 4
 5      boolean result;
 6      int     intValue;
 7      double  doubleValue;
 8      String  description;
 9
10⊖     public boolean isResult() {
11          return result;
12      }
13⊖     public void setResult(boolean result) {
14          this.result = result;
15      }
16⊖     public int getIntValue() {
17          return intValue;
18      }
19⊖     public void setIntValue(int intValue) {
20          this.intValue = intValue;
21      }
22⊖     public double getDoubleValue() {
```

Static

Static Fields

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

```java
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 Methos

- 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?

```java
@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);

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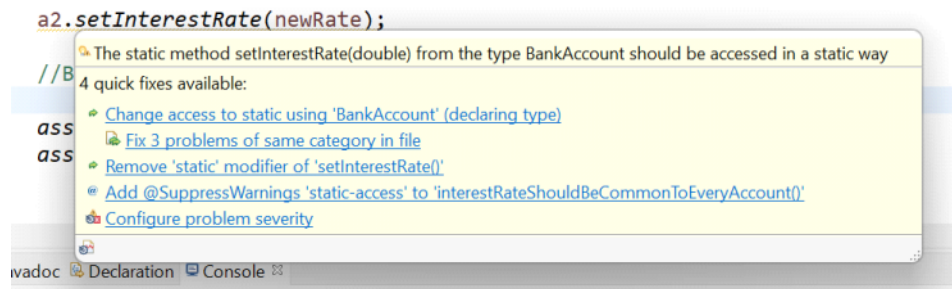
- this code appears to be chaning the interest rate for a particular object "a2"
- Actually it is changing for everyone which is not clear by looking at the code
- static will allow you to call this method using class reference

```
//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 recommedns that static methods should be accessed only using class referenc eand NOT using object reference.

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

The static method setInterestRate(double) from the type BankAccount should be accessed in a static way

//B 4 quick fixes available:

- Change access to static using 'BankAccount' (declaring type)
    - Fix 3 problems of same category in file
- Remove 'static' modifier of 'setInterestRate()'
- Add @SuppressWarnings 'static-access' to 'interestRateShouldBeCommonToEveryAccount()'
- Configure problem severity

vadoc   Declaration   Console

Do I really need static?

- 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 Oreinted.

- interestRate and lastId isn't owned by account object
  - who owns them?

```java
public class BankAccount {

    int accountNumber;
    String name;
    String password;
    double balance;

    static double interestRate;

    static int lastId=0;
```

```java
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);
    }

}
```

```java
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);
```

Test Elements

Wednesday, February 8, 2023   11:02 AM

AAA —> Arrange Act Assert

Arrange (Setup)
- setup the intial 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

- 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 initally 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 acknolwdging 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 you 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

- Inhertiance 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 Veichle
- 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 refered 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 extend Animal{

    void hunt();

}
```

```
public void doTaskB(){

    }
}
```

- Now

```
                                    }

Animal myFavAnimal = new Tiger()
```

```
┌─────────────────────────┐
│            X            │
├─────────────────────────┤
│        -  a : int        │
├─────────────────────────┤
│       + doTaskA()        │
└─────────────────────────┘
             △
             │
┌─────────────────────────┐
│            Y            │
├─────────────────────────┤
│        -  b : int        │
├─────────────────────────┤
│       + doTaskB()        │
└─────────────────────────┘
```
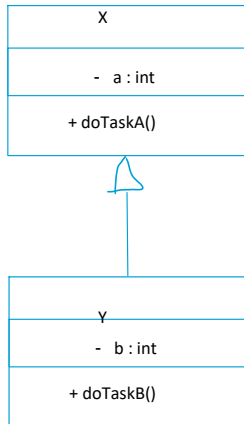
Inheritance is about creating a relationship: Is A Type Of

- a class Y can extend any class X
- X and Y can be anything
  - Airplane extends Cow
  - Cycle extends Circle
  - RubberDuck extends RealDuck
  - 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{
```

```java
        //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 Strin hunt(){
        return "Hunt's it's pray";
    }
}
```
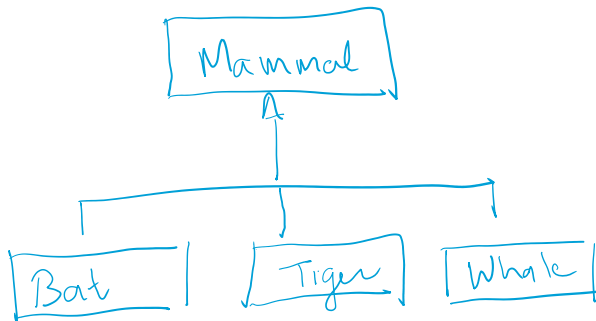
- Starting Java 6, we have an annotation to mark a method that overrides another method from the super class.
- This annotation is optional (as it was introduced late) but recommended.
- Advantage
  - If you write an annotation of a method that doesn't have a correspondence in the super class it will give you a compile time error

```java
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

```java
Mammal [] mammals = {

    new Tiger(),
    new Bat(),
    new Whale()

}
```

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

```java
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 olden compiler driven langauges 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 behaing is**
  - Polymorhphism

Polymorphism

```
Mammal mammal = getARandomMammal();


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

Inhertiance 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 memebers) 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 overriden 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 Sring 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" ;}
}
```

```java
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

```java
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

```java
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

```java
((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 iternation
- crashes while tyring to typecast a Horse into a Tiger
    - This is a runtime error

Always typecast when you are sure it wouldn't crash

```java
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());
   }

}
```
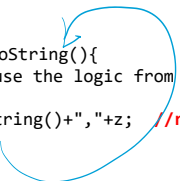
Partial Overriding or Partial Modification

```
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 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 overrride or modification
  - we either override or we don't
- Once we override, the subclass method will hide the superclass implementation
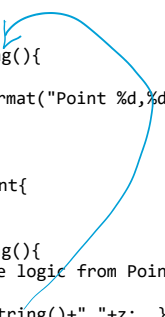
we can access the super class version of method using super reference

```
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 initalization
  - 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 overriden
  - 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

- 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 extens 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
  - It 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 initalize 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

```java
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

```java
public Point3d( int x, int y, int z) {

    this.z=z;
    super(x,y); //MUST BE FIRST STATEMENT


}
```

Polymorphism

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

Class Oriented vs Object Oriented

Tuesday, February 14, 2023   8:24 AM

//Approach A

```
Animal tiger = new Animal (AnimalType.Tiger);

Animal eagle =new Animal (AnimalType.Eagle);

Animal snake = new Animal(AnimalType.Snake);
Animal crocodile =new Animal(AnimalType.Crocodile);

Animal animals[]={tiger,eagle,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");
                break;
            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

```
Tiger tiger =new Tiger();

Eagle eagle = new Eagle();

Snake snake =new Snake();


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



    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{
```

```java
        }


// client code

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


…

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


public void printMammals(Animal[] animals){

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

    }

}
```

```java
public void printMammals(Animal[] animals){

}
```

Animal Hierarchy

Tuesday, February 14, 2023   9:37 AM

```
 3  public class Animal {
 4
 5
 6⊖      public String eat() {
 7          return this+" eats something";
 8      }
 9
10⊖      public String move() {
11          return this+" moves somehow";
12      }
13
14⊖      public String breed() {
15
16          return this+" breads somehoe";
17      }
18
19⊖      public String toString() {
20          return getClass().getSimpleName();
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 <mark>sufficient specific information available to create instances of this class</mark>
    - Actual instances will be created for the sub class
    - <mark>This class is meant only to be a super class</mark>

```
 2
 3  public class Animal {
 4
 5
 6⊖      public abstract String eat();
 7                                 ┌──────────────────────────┐
 8⊖      public String move() {     │ ⊘ Remove 'abstract' modifier │
 9          return this+" moves     │ ⊘ Make type 'Animal' abstract │
 0      }                          └──────────────────────────┘
 1
 2⊖      public String breed() {
```

- If even one method is abstract, the class must be abstract

- When class becomes abstract you can't instantiate it

```
25      new Animal(),
26      new  ┌ ⊗ Cannot instantiate the type Animal ┐
27      new  └                                      ┘
28      new Dog(),
```

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

```
ublic 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 overriden 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      }
10      public St
11          return
12      }
```

The type Mammal must implement the inherited abstract method Animal.eat()
2 quick fixes available:
- Add unimplemented methods
- Make type 'Mammal' abstract

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
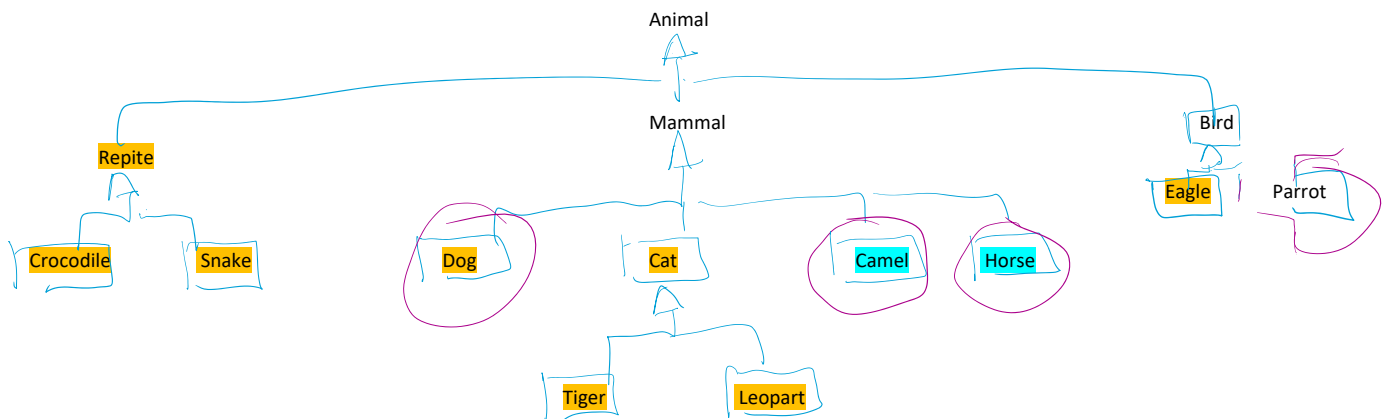
What is the relationship between animals marked Orange?

- 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

- 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 explictly
      - but you cant use other modifiers like protected
    - There is no concrete or non-public member
  - You implement and 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?

```java
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

Inteface Implementation

```
 4
 5 public class Parrot extends Bird implements Hunter{
 6
 7    @Override              The type Parrot must implement the inherited abstract method Hunter.hunt()
 8    public St              2 quick fixes available:
 9        // TO               • Add unimplemented methods
10        retur              • Make type 'Parrot' abstract
11    }
```

```java
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

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

- A Rideable is one you Ride on

```java
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

```java
public interface Pet {

}
```

Marker Interface

- 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
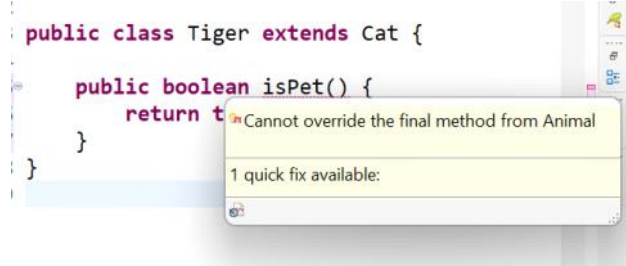- It gives me an ability to test for "instanceof"

Final Method/Class

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

```java
public abstract class Animal {

    public Animal() {☐

    public abstract String eat();

    public abstract String move();

    public abstract String breed();

    public final boolean isPet() {
        return this instanceof Pet;
    }
}
```

```java
public class Tiger extends Cat {

    public boolean isPet() {
        return t☐ Cannot override the final method from Animal

                 1 quick fix available:
}
```

Final Class

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

```java
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
```

```java
interface BankingStatus{

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

}


class BankingStatus{

    public final String succcess="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 or all Java classes
    - predfined
    - 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 instance of 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(){…}
```

```java
        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

```java
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 Hierarcy

```
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

- java.lang
  - The most important of all Java Packages
  - It contains most important classes related to any Java Proramming 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 utilitiy functions not as important as java.lang
  - we need to explcitly 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
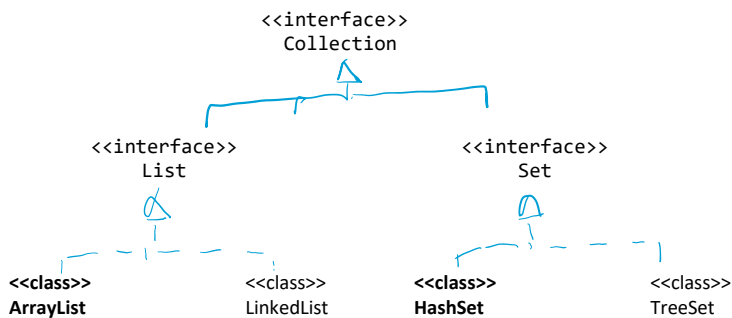- java.awt
  - For devloping java desktop application

Java Collection Classes

Tuesday, February 14, 2023   12:02 PM

- Java provides a set of dynamic collection classes
- Think of a 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 imporoves after Java 6

Pre Java 6

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

```
                    <<interface>>
                     Collection
```

```
    <<interface>>                    <<interface>>
        List                             Set
```

```
<<class>>            <<class>>    <<class>>            <<class>>
ArrayList            LinkedList   HashSet              TreeSet
```

```
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 unqiue values
}
```

- ArrayList
  - It is a dynamic array
  - It is expanded in an optimized way
  - One of the most popular collection
  - Memory is internally continous
  - good to access random values programmetically

- LinkedList
  - Uses double linked list algorithm
  - data is stored in non-contgous nodes
  - good choice if you need to insert values in between collection

- TreeSet
  - stores unqiue 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 unqiue set of values using hashing algorithm
  - values are not sorted
  - It is optimized for fast search of data inside the collection

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);
}
```
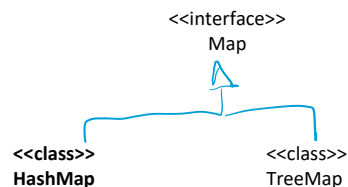
A collection that is not a Collection

//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 a 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 premitive types
- To use premitive type you need to use Object wrappers

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

```java
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 storae 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 symatically perpfectly
```

• 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 paritcular 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

}
```

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 premitive

```
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 Deparment
  - 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 explcit syntax

```
static ArrayList getHunters(Animal [] animals) {

    ArrayList hunters=new ArrayList();
```

ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized

5 quick fixes available:

- Add type arguments to 'ArrayList'
  - Fix 4 problems of same category in file
- Infer Generic Type Arguments...
- Add @SuppressWarnings 'rawtypes' to 'hunters'
- Add @SuppressWarnings 'rawtypes' to 'getHunters()'
- Configure problem severity

```
}
```