Java Basics- Data types

Wednesday, June 11, 2025 3:05 PM

```
byte c= 30;
//  c= c * 5; cannot convert from int to byte
c= (byte) (c*5);
```

1. The expression x * 5 involves multiplying a byte (which is 3) by an int (5). The result of this multiplication is an int, specifically 15.
2. When you try to assign this int result to a byte, you need to explicitly cast it back to byte

Signed Types

For signed types, the range is calculated as follows:

- **Formula**:

$$\text{Range} = -2^{(n-1)} \text{ to } 2^{(n-1)} - 1$$

Explanation:

1. **Total Combinations**: With n bits, there are 2^n total combinations of 0s and 1s.
2. **Sign Bit**: One bit is used to represent the sign (positive or negative). This leaves $n - 1$ bits for the magnitude of the number.
3. **Negative Range**: The smallest value is represented by all bits being 0 except the sign bit, which is 1, resulting in $-2^{(n-1)}$.
4. **Positive Range**: The largest positive value is represented by all bits being 1 except the sign bit, which results in $2^{(n-1)} - 1$.

Unsigned Types

For unsigned types, the range is calculated as follows:

- **Formula**:

$$\text{Range} = 0 \text{ to } 2^n - 1$$

Explanation:

1. **Total Combinations**: With n bits, there are still 2^n total combinations.
2. **No Sign Bit**: All bits are used for the magnitude since there is no sign bit.
3. **Minimum Value**: The smallest value is 0 (all bits are 0).
4. **Maximum Value**: The largest value is when all bits are 1, which is $2^n - 1$.

```
byte b1 = 127; // range -128 to 127
System.out.println(Byte.MAX_VALUE); //127

float f = 2E7f; // scientific notation, 2 * 10^7
//OR
float f2= 2e7F; // scientific notation, 2 * 10^7
```

Rules for naming variables:-

- Names can also begin with $ and _

_____

division (9/5) is performed as integer division, which results in 1 instead of 1.8.
To fix this, you should use 9.0/5 to ensure floating-point division.

Wrapper class in java

Friday, June 13, 2025 6:34 PM

Wrapper classes in java

Wrapper classes are final and immutable.

- Thread Safety: Multiple threads can safely access and share the same wrapper object (e.g., an Integer object) simultaneously without the need for synchronization.
- Core behavior of the wrapper object cannot be modified. (final class can't be inherited)
- Any operation that seems to change the object actually returns a **brand new wrapper object**, leaving **the original object untouched**.

== with respect to Integer wrapper class

```
        Integer int1 = 100;
        Integer int2 = int1;
        System.out.println(int1 == int2);     //true
       //compares the references, not the values

        Integer int3 = 100;
        System.out.println(int1 == int3); //true

        //In Java, Integer objects within the range of -128 to 127 are cached by the JVM.
        // This means that when you assign Integer int3 = 100;, it refers to the same cached object as int1 and int2 (when their value is also 100).
        // Therefore, int1, int2, and int3 all point to the same object in memory for the value 100.


        Integer int4 = 200;
        Integer int5 = int4;
        System.out.println(int4 == int5); //true

        Integer int6 = 200;
        System.out.println(int5 == int6); //false


        int1 = 150; //a new Integer object is created because wrapper classes are immutable
        System.out.println(int1 == int2); //false

    }
```

Integer:
intValue() method - Returns the value of this Integer as an int.
compareTo() method - returns 0, -1, or 1 based on the difference

---

Questions and answers based on Integer wrapper class:

Basic Questions:
1. What is the purpose of the `Integer` wrapper class in Java?
2. How does autoboxing and unboxing work with the `Integer` class?
   - autoboxing: primitive value to wrapper class
3. What is the range of values for the `Integer` class in Java?
   - same as the range of the int primitive type
4. What is the difference between `int` and `Integer` in Java?

Memory Management:
5. How does the Integer Cache work in Java? What is its range?
6. What happens when you assign a value outside the Integer Cache range?
   - Yes, when you assign a value outside the Integer Cache range (which is -128 to 127), a new Integer object is created

7. Explain the immutability of the `Integer` class.
   - The value of an Integer object is fixed once it is created.
     Any operation that appears to modify the value of an Integer actually creates a new Integer object.

Comparison:
8. What is the difference between `==` and `.equals()` when comparing `Integer` objects?
   - When used with Objects:
     == compares references
     equals() compares the values

   - When used with primitives == compares values

9. How does the `compareTo()` method work in the `Integer` class? What are its possible return values?

Methods:
10. What is the purpose of the `Integer.parseInt()` method? How is it different from `valueOf()`?
    - String to int conversion

```
//Parsing a decimal string
int number1 = Integer.parseInt("123");
System.out.println(number1);

//Parsing a binary string
int number2 = Integer.parseInt("1010", 2);
System.out.println(number2);
```

parseInt(String s): Returns a primitive int.
valueOf(String s): Returns an Integer object (wrapper class).

11. How does the `Integer.toString()` method work?
12. What is the use of `Integer.sum()`, `Integer.max()`, and `Integer.min()` methods?

Advanced:
13. How does the `Integer.reverseBytes()` method work?
14. What is the purpose of the `Integer.signum()` method?
15. Explain the `Integer.bitCount()` method and its use.

Practical Scenarios:
16. Why is the `Integer` class declared as `final`?
17. What happens when you modify the value of an `Integer` object?
18. How does the `Integer` class handle binary, octal, and hexadecimal conversions?

Miscellaneous:
19. Can you inherit the `Integer` class? Why or why not?

20. What are the differences between `Integer` and other wrapper classes like `Double` or `Long`?
    - Integer: Caches values between -128 and 127 for autoboxing.
      Double: Does not cache values.
      Long: Caches values between -128 and 127 for autoboxing.

Wrapper classes in Java are designed to wrap primitive data types (like int, char, boolean) into objects. String, while also an object, represents a sequence of characters and is not a wrapper for any primitive type.

```
Integer int13 = null;
System.out.println(int13); //null

String s1 = Integer.toString(87356);
        // NullPointerException- String s2 = int13.toString();

String s2 = String.valueOf(int13);    // converts null to "null"
```
---

char - primitive
Character- wrapper

final

Sunday, October 19, 2025 12:50 AM

- **final used on a class**

When a class is declared as final, it means the class cannot be extended or subclassed.

- **final used on a reference of a class**

When a variable is declared as final and it holds a reference to an object, it means the variable cannot be reassigned to point to a different object.

Example (using final on the reference of inbuilt class in java):-

final StringBuilder myBuilder = new StringBuilder("Initial"); // This is allowed because we are modifying the object's state, not the reference.

myBuilder.append(" String"); // compile-time error because we are trying to reassign the 'final' reference.

myBuilder.insert(7, "Modified "); // This is allowed because we are not reassigning the variable 'myBuilder'.

Practice from- Codechef

Wednesday, June 18, 2025 11:39 AM

1. System.out.println(7.0/2); // 3.5

2. "\n" - for next line

3. Strings topic->
charAt()
replace()
concat()

4. Convert String to character array, so that we can modify the characters

5. To input a character->
scanner.next().charAt(0)

6.

```
Scanner scanner = new Scanner(System.in);
char first, second;
first = scanner.next().charAt(0);
second = scanner.next().charAt(0);
System.out.println(first + second);  // prints the sum of ASCII values of characters
scanner.close();
```

7.

```
String[] words = str.split(" ");
```

8.

```
char ch = 'a';
String str = String.valueOf(ch);
```

9. It is possible to override equals() method in our custom class and provide our implementation.

10. final keyword makes the reference immutable.
String is immutable, but String can create a new object and point to a new memory location with new object.
But, final keyword prohibits the variable from pointing to a new memory location.

11.
int x = 10;
System.out.println("x = "+ x + 5);   // x = 105

When an integer is added to a string, the integer is converted to its string representation and then concatenated.

12. String str = null;
null represents the absence of an object.

13.
next() - next() will read characters until it encounters a whitespace

nextLine(): This method reads the entire line of input until it encounters a newline character (\n). It consumes the newline character and returns the string containing all characters up to, but not including, the newline.

NAME WILL NOT BE TAKEN BECAUSE nextLine() will consume the newline character
int age = sc.nextInt();
String name = sc.nextLine();

SOLUTION:-
int age = sc.nextInt();
sc.nextLine();
String name = sc.nextLine();

Typecasting

Tuesday, June 17, 2025 11:58 AM

```
        //Upcasting
        System.out.println("5. Upcasting");
        Payment payment = new CreditCardPayment();
        payment.processPayment();
        // cannot call credit card specific method

        //Down casting
        System.out.println("6. Down casting");
        if(payment instanceof CreditCardPayment){
            CreditCardPayment creditCardPayment = (CreditCardPayment) payment;
            creditCardPayment.creditCardSpecificMethod();
        }


class Payment {
        void processPayment() {
        System.out.println("Processing payment... ");
        }
    }

    class CreditCardPayment extends Payment {
        void processPayment() {
        System.out.println("Processing credit card payment... ");
        }

        void creditCardSpecificMethod() {
        System.out.println("Inside credit card specific method");
        }
    }

-------------------------------------------------------------


class A {
    static void print() {
        System.out.println("A");
    }
}

class B extends A {
    static void print() {
```

```
        System.out.println("B");
    }
}

public class Test {
    public static void main(String[] args) {
        A obj = new B();
        obj.print();  // A
    }
}
```

static methods are not subject to runtime polymorphism.
Here, the method call is resolved at compile time based on the reference type.

_____
Widening- implicit casting
Narrowing- explicit casting
_____

Questions and answers->

How does typecasting enable polymorphism in Java?
- allowing a subclass object to be treated as an instance of its superclass

What is a ClassCastException? When does it occur?
- ClassCastException - occurs at runtime if the object being cast is not of the expected type

How can you avoid ClassCastException during downcasting?
- use instanceOf

Can you explain the difference between compile-time and runtime typecasting errors?
->

```
List<String> list = new ArrayList<>();
list.add("Hello");
// No need for casting
String str = list.get(0);

Without generics, you would need to cast the object:-
List list = new ArrayList();
list.add("Hello");
String str = (String) list.get(0); // Explicit cast required
```

How does typecasting work with generics in Java?

Is it possible to cast between unrelated classes in Java? Why or why not?
_____

Widening:-

```
long num = Long.MAX_VALUE;  // 9223372036854775807
float f = num;
System.out.println(f);  // 9.223372E18
```

long (8 bytes): It is a 64-bit integer type that can represent whole numbers in the range of -2^63 to 2^63 - 1.
float (4 bytes): It is a 32-bit floating-point type that can represent a much larger range of values (approximately ±3.4 × 10^38) but with less precision.
Key Points:
Range: A float can represent numbers outside the range of a long, so converting a long to a float is safe in terms of range. This is why it is considered a widening conversion.

Java allows implicit conversion from long to float because the range of float can accommodate the values of long, but precision may be lost.

_____

Static variable

Sunday, June 22, 2025 11:42 PM

1. Why can't static variable be declared in a static context?

Static variables must be declared at the class level.
We can initialize the static variables inside the static block.

Conditional statements

Monday, June 23, 2025 12:06 AM

1)
if, else if, else, switch

2) Logical Operators- AND, OR, NOT

Logical Or - ||
This operator doesn't check the second condition if the first one is true. The second condition is checked only and only if the first condition is false.

3) **XOR**- returns true only if condition 1 and condition 2 are different.
A ^ B =
A XOR B equals (NOT A AND B) OR (A AND NOT B)

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Example:-
Two-way or three-way light switch

Light ON if the two switches are in **different** positions.

---

^ operator in java - XOR operations

int x = 4;
int y = 4;

System.out.println(x ^ y); // 0

Any number XORed with itself is 0.
Any number XORed with 0 remains unchanged.

Find unique element in array of integers where the integers occur twice except one unique integer

The XOR method is specifically designed to find a single unique element in an array **where all other elements appear exactly twice**.

```
int arr[] = {4, 3, 4, 5, 3};

int unique = 0;

// XOR all elements in the array
for (int num : arr) {
   unique ^= num; // XOR operation
}
```

O(n) time complexity.

Explanation of the logic:-

4 ^ 3 :-

 100
^(XOR)
 011

 Answer:- 111 (7)

 7 ^ 4 :- 3

 So,
 4 ^ 3 ^ 4 :- 3

 Similarly for the array:-

4 ^ 3 ^ 4 ^ 5 ^ 3 :- 5
(4 and 5 will be cancelled)

Functions/Methods

Saturday, July 5, 2025 10:10 PM

Code reusability

- The bytecode of methods (the actual instructions) and static variables reside in the **Method Area** (Metaspace). This is the blueprint or definition of the method.
- When a method is actively being executed, its local data (parameters and local variables) is stored in a stack frame on the thread's Stack.
- Objects that methods operate on, and their instance variables, are stored in the Heap.

String

Sunday, July 6, 2025 12:46 AM

Methods:-

⊙ substring
- substring(int beginIndex)
  - from beginIndex till the end of the string
- substring(int beginIndex, int endIndex)
  - up to but not including endIndex

⊙ compareTo

str1.compareTo(str2);
= 0 (same strings)
> 0 (if str1 is lexicographically greater than str2)
< 0 (if str1 is lexicographically lesser than str2)

unicode value based comparison

'A' < 'a'
('a' is lexicographically greater than 'A')

⊙ toLowerCase

⊙ toCharArray

```
String a = sc.next().toLowerCase();
String b = sc.next().toLowerCase();

char[] charArray1 = a.toCharArray();
char[] charArray2 = b.toCharArray();

Arrays.sort(charArray1);
Arrays.sort(charArray2);

a = new String(charArray1);
```

```
        b = new String(charArray2);
```

Split string with multiple delimiters

```
String[] splitArr = str.split("[!,?._'@ ]");
```

StringBuilder

Monday, July 7, 2025 3:03 PM

```
StringBuilder sb1 = new StringBuilder(substr2);
sb1.reverse();  // StringBuilder object is mutable

String str = String.valueOf(sb1)
```

Basic DSA

Monday, July 7, 2025 4:07 PM

Array, String, List

Linear Search
Binary Search

Bubble sort
Selection sort
Insertion sort
Merge sort
Quick sort

Arithmetic operations

A)

Bubble sort- Compares two adjacent elements and swaps them until they are in order

Optimized bubble sort:-
If the array is already sorted-> Put a flag in the code.
break the execution if sorted.

B) Selection sort in my words:-

At first-> Consider 1st element as a part of sorted array.
Scan the remaining array to find the element smaller than the 1st element and if you find it, swap it with the smallest element.

Next iteration-> pointer i on 2nd element.
Find the element from the remaining array smaller than this 2nd element and swap it with the element at pointer i.

C) Insertion sort in my words:-

sorted array and unsorted array partitions
1st element is sorted.

Take one element from unsorted array and put it at the right place in sorted array by swapping the elements with it that are greater.

D)

Constructor

Thursday, June 26, 2025 7:20 PM

A)

In Java, super() or this() must be the very first statement inside a constructor. You can't have any logic (like variable declarations or assignments) before them.

1. The parent class must be initialized first, and
2. The call to the parent constructor (super()) needs to happen before the current object (this) is fully formed.

B)

Date Time

Sunday, June 29, 2025 7:35 PM

java.time package to work with date and time API

```
LocalDate joiningDate = LocalDate.of(2022, 9, 30);
        LocalDate presentDate = LocalDate.now();

        long yearsOfService = Period.between(joiningDate,
presentDate).getYears();
        System.out.println(yearsOfService);

        long yearsOfService2 =
ChronoUnit.YEARS.between(joiningDate, presentDate);
        System.out.println(yearsOfService2);
```

Question-Ans

Friday, July 4, 2025 4:23 PM

1. Perfect square logic

double squareRoot = Math.sqrt(num);
int root = (int) squareRoot;

if((squareRoot - root == 0) -> then the number is perfect square

2D ArrayList

Thursday, July 10, 2025 12:11 PM

```
ArrayList<ArrayList<Integer>> twoDList = new ArrayList<>();

ArrayList<Integer> row1 = new ArrayList<>();
row1.add(1);
row1.add(0);
row1.add(-2);
twoDList.add(row1);

ArrayList<Integer> row2 = new ArrayList<>();
row2.add(-56);
row2.add(-3);
row2.add(67);
twoDList.add(row2);

int rowSize = twoDList.size();

int colSize = twoDList.get(0).size();

for(int i=0; i<rowSize; i++){
   for(int j=0; j<colSize; j++){
      System.out.print(twoDList.get(i).get(j) +" ");
   }
   System.out.println();
}
```

Collections

Monday, August 11, 2025 5:14 PM

1. List.copyOf()
   - creates an unmodifiable list
   - It cannot contain null elements

Abstract class Vs. Interface

Thursday, August 28, 2025 5:08 PM

1. When to use abstract class and interface- with example:

1. **Abstract Class: Shared Code Among Related Classes**

Situation:
You have several closely related classes that should share some state (fields/variables) and some common behavior (methods), but also should provide their own specific implementations for other behaviors.

Example: Animal Hierarchy
Suppose you have different animals (`Cat`, `Dog`) that all have a `name` and can `eat()`, but each makes a different sound.

```java
abstract class Animal {
    String name;

    Animal(String name) {
        this.name = name;
    }

    void eat() {
        System.out.println(name + " is eating.");
    }

    abstract void makeSound();
}

class Dog extends Animal {
    Dog(String name) {
        super(name);
    }
    void makeSound() {
        System.out.println("Woof!");
    }
}
```

```
class Cat extends Animal {
   Cat(String name) {
      super(name);
   }
   void makeSound() {
      System.out.println("Meow!");
   }
}

public class Main {
   public static void main(String[] args) {
      Animal a = new Dog("Max");
      a.eat();            // uses concrete code from abstract superclass
      a.makeSound();      // uses subclass-specific code
   }
}
```

Use abstract class:
- Because all animals have a name and the ability to eat.
- You want to share the `eat()` implementation.
- You want subclasses to provide their own `makeSound()`.

---

2. **Interface: Define Capability Across Unrelated Classes**

Situation:
You want to state that any class (even if not closely related) can have a certain capability, such as being able to be driven or flown.

Example: Drivable/ Flyable Vehicles and Animals
Both a `Car` and an `Airplane` can be “driven,” and a `Bird` and a `Plane` can “fly,” but there’s no direct relationship between them.

```java
interface Drivable {
   void drive();
}

interface Flyable {
   void fly();
}
```

```
class Car implements Drivable {
   public void drive() {
      System.out.println("Car is being driven.");
   }
}

class Airplane implements Drivable, Flyable {
   public void drive() {
      System.out.println("Airplane is taxiing.");
   }
   public void fly() {
      System.out.println("Airplane is flying.");
   }
}

class Bird implements Flyable {
   public void fly() {
      System.out.println("Bird is flying.");
   }
}

public class Main {
   public static void main(String[] args) {
      Drivable d = new Car();
      d.drive();

      Flyable f = new Airplane();
      f.fly();
   }
}
```

Use interface:
- To specify that unrelated classes (like `Car` and `Airplane`, or `Bird` and `Plane`) share certain behaviors/capabilities, even without common ancestry.
- Enables **multiple inheritance** of abilities (e.g., `Airplane` is both `Drivable` and `Flyable`).

Use Case	Prefer Abstract Class	Prefer Interface
Classes are related by "is-a"	✔	
Want to share code/fields	✔	
Provide default method implementations	✔	Java 8+ default methods only
Define capabilities for any class		✔
Enable multiple inheritance of types		✔
No shared implementation needed		✔

In summary:
- Use abstract classes for shared code & closely related classes.
- Use interfaces for defining capabilities across unrelated classes or when you need multiple inheritance of behavior.

Threads and Multithreading- Basics

Saturday, August 30, 2025 8:35 PM

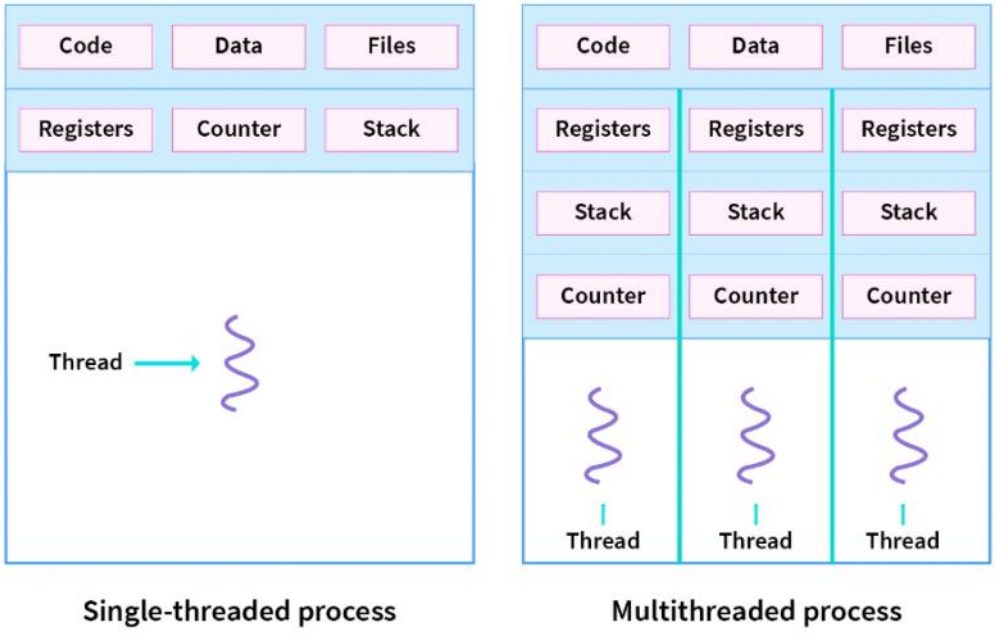
My questions- answered by Schaeffler Chatbot

Article on- 'Threads in Operating System'
https://www.scaler.com/topics/operating-system/threads-in-operating-system/

Threads

Threads *can* run truly in parallel (at the exact same time) on multi-core processors, but on a single core, they run concurrently by rapidly switching (context switching) between them

Each thread has its own program counter, stack, and set of registers. However, the threads of a single process might share the same code and data/file.



A kernel is the core, central program of an operating system (OS), acting as a crucial bridge between software applications and hardware, managing essential system resources like CPU, memory, and I/O, and handling fundamental tasks such as process scheduling, memory allocation, and device communication to ensure smooth, efficient system operation
---
User level thread-

- User-level threads are implemented using user-level libraries and the OS does not recognize these threads.
- Cannot utilize multiple CPU cores (true parallelism). The Kernel only sees one execution flow (the process) and schedules that one

process to run on a single core at a time.
- Context switching in user-level threads is faster.
- ULT operations (like context switching) do not require kernel calls or hardware-level interrupts.

Kernel level thread-
- Can utilize multiple CPU cores simultaneously, as the Kernel can schedule different KLTs to run on different cores.
- KLT operations (like scheduling or blocking) always require a system call to the Kernel.

---

1. Process
A **process** is an independent program running in its own memory space. For example, when you open Chrome and Notepad, each is a separate process.

2. Thread
A **thread** is the smallest unit of execution within a process. A process can have multiple threads (called multithreading) that share the same memory.

3. Process States, Thread States
- **Process states**: Created, Ready, Running, Waiting (Blocked), Terminated.
- **Thread states**: New, Runnable, Running, Waiting, Timed Waiting, Terminated.
  (Names may slightly vary in different languages/systems.)

4. Context Switching
Switching the CPU from one process or thread to another. It saves the state of the old process/thread and loads the state for the new one.

5. Concurrency
Ability to deal with multiple tasks at the same time, but **not necessarily executing them simultaneously**. For example, taking turns to use the CPU.

6. Parallelism
Actually **executing multiple tasks at the exact same time**, usually on multiple CPU cores.

7. By default single thread?
Yes. Most programs start with a single thread: the "main" thread.

8. When to use multithreading?
- When tasks are independent and can be run at the same time.
- For tasks like downloading files, handling multiple clients, or making UI more responsive.

9. How do we know whether a method could be accessed by multiple threads?
- If multiple threads can call the same method **at the same time**, especially when they share data.
- Any **shared resource** (object/variable) in a multithreaded program should be considered.
- Look for code that is run inside "run()" methods or thread pools, or that is accessed by multiple user sessions (like in servers).

10. What is an asynchronous method?
A method that **doesn't block** the caller. It starts a task and lets the rest

of the program continue running while the task finishes in the background.

11. What is synchronization?

A way to **control the access** of multiple threads to shared resources, so only one thread can access the critical section at a time. Prevents problems like corrupted data.

12. What are race conditions?

When two or more threads try to **change the same data at the same time** without proper synchronization, causing unpredictable or incorrect results.

---

13. **When should I worry about multithreading as a backend developer**?

- When I share data among multiple users/requests
- Don't use normal (non-thread-safe) changing variables (like counters, lists) in your Spring controllers or services, because many users can access them at the same time and it can cause problems.
- If different users or parts of your program might change the same data at once, be careful—otherwise, you can get wrong or mixed-up data.

---

14. **synchronized**

```
public class ThreadSync extends Thread{
    public void run(){
        System.out.println("Not synced"); // this statement can be accessed by multiple threads simultaneously

        synchronized (ThreadSync.class){ // only this block will be synced
            for(int i=1; i<=10; i++){
                System.out.println(i);
            }
        }

       //synchronized block ensures that only one thread at a time can execute the block synchronized on ThreadSync.class.

        // Non-synchronized statements (like System.out.println("Not synced");) can still be executed by other threads concurrently

    }
}

class SyncTest{
    public static void main(String[] args) {
        ThreadSync threadSync1 = new ThreadSync();
        threadSync1.start();

        ThreadSync threadSync2 = new ThreadSync();
        threadSync2.start();
    }
}
```

---

15. **Thread Pool**

- a collection of pre-created, reusable threads that are kept ready to perform tasks.
- No need to create new thread every time- which is costly in terms of memory and CPU

- a thread pool maintains a fixed number of threads. When a task is submitted:

If a thread is free, it immediately picks up the task and runs it.
If all threads are busy, the task waits in a queue until a thread becomes available.
After finishing a task, the thread does not die. It goes back to the pool and waits for the next task.
---

16. **Thread priority**

Thread priority in Java is a numerical value from 1 (lowest) to 10 (highest)

The thread scheduler uses priority to decide which thread to run, but the actual execution depends on the operating system and JVM implementation, so priority is a hint, not a guarantee

Thread.MIN_PRIORITY (1),
Thread.NORM_PRIORITY (5),
Thread.MAX_PRIORITY (10)

I/O Streams and file handling

Thursday, September 4, 2025 12:33 PM

Scanner - tool or blueprint for reading input

Scanner sc = new Scanner(System.in)
- Get the data from keyboard

System.in - Standard Input Stream

_____

I/O Streams and File Handling

I/O streams are the fundamental concept for moving data in and out of a Java program. File handling is a specific application of this concept, where the source or destination of the data stream is a file.

- **java.io**: This is the traditional package for Java's I/O operations. It contains classes like FileInputStream and FileOutputStream for reading and writing data to files in the form of **bytes**, and classes like FileReader and FileWriter for handling **characters**.
- **java.nio**: This package, introduced in Java 1.4, provides a newer, more efficient way to perform I/O. It stands for **New I/O**. It's often referred to as "NIO.2" with later updates. While java.io uses streams, java.nio is **buffer-oriented** and uses **channels**. This allows for non-blocking I/O, which is particularly useful for high-performance applications like network programming, but is also used for advanced file handling.
- **FileInputStream and FileOutputStream**: These are the workhorses of byte-based file I/O.
  - FileInputStream is an **input stream** used to read a stream of raw bytes from a file.
  - FileOutputStream is an **output stream** used to write a stream of raw bytes to a file. These are ideal for handling binary data like images, audio, or video files.
- **FileReader and FileWriter**: These are similar to the above but are specifically designed for character-based data, like text files.
  - FileReader reads a stream of characters from a file.
  - FileWriter writes a stream of characters to a file. They automatically handle the conversion from bytes to characters based on the default character encoding.
- **BufferedReader and BufferedWriter**: These classes are **wrappers** that enhance the performance of other I/O streams. They read or write data into a **buffer** in memory, reducing the number of times the program has to access the physical disk.
  - BufferedReader wraps an input stream (like FileReader) to read text more efficiently, often line by line.
  - BufferedWriter wraps an output stream (like FileWriter) to write text more efficiently.
- **RandomAccessFile**: This class is unique because it's not a stream. Instead, it allows for **random access** to a file, which means you can move a file pointer to any location and read or write from there. This is different from streams, which read or write data sequentially.

_____

```
import java.io.InputStream;
import java.nio.charset.StandardCharsets;

import org.apache.commons.io.IOUtils;
import org.springframework.core.io.DefaultResourceLoader;
```

```
import org.springframework.core.io.Resource;
import org.springframework.core.io.ResourceLoader;

public class FileUtil {

   public static String getResourceFileContent(String resourcePath) {
      ResourceLoader resourceLoader = new DefaultResourceLoader();
      Resource resource = resourceLoader.getResource("classpath:" + resourcePath);
      String content;
      try {
         InputStream inputStream = resource.getInputStream();
         content = IOUtils.toString(inputStream, StandardCharsets.UTF_8);
         inputStream.close();
      } catch (Exception e) {
         throw new RuntimeException("Ops, reading resource file <" + resourcePath + "> has failed", e);
      }
      return content;
   }
}
```

1) **ZonedDateTIme** class in java

ZonedDateTime is an immutable object representing a date-time along with the time zone. This class stores all date and time fields. This class stores time to a precision of nanoseconds and a time-zone, with a zone Offset used to handle local date-times. For example, the value “2nd October 2011 at 14:45.30.123456789 +05:30 in the Asia/Kolkata time-zone” can be stored in a ZonedDateTime. This class is also used to convert the local time-line of LocalDateTime to the instant time-line of Instant.

---

2) **Locale** class in java

---

3) enum

```
public enum PspaLinkRelationTypes {

   CREATE_PSPA("create-pspa"),
   GET_PSPA("get-pspa"),
   UPDATE_PSPA("update-pspa");

   private final String relationType;

   PspaLinkRelationTypes(String relationType) {
      this.relationType = relationType;
   }

   public String get() {
      return relationType;
   }
}
```

In order to use it, import the java file wherever neeeded.
CREATE_PSPA.get() :- the result is create-pspa

---

4) StringJoiner in java

StringJoiner is a class in Java that allows the programmer to construct a sequence of characters separated by a delimiter and optionally starting with a prefix and ending with a suffix. It was introduced in Java 8 and provides a convenient way to join strings in an efficient and concise manner. The StringJoiner class provides methods such as add() to add elements to the sequence, setEmptyValue() to set a default value in case there are no elements to join, and toString() to return the final concatenated string. It is commonly used in generating CSV and JSON files.

```
import java.util.StringJoiner;

public class Example {
   public static void main(String[] args) {
      StringJoiner joiner = new StringJoiner(",", "[", "]");
      joiner.add("apple");
```

```
        joiner.add("banana");
        joiner.add("cucumber");
        String joinedString = joiner.toString();
        System.out.println(joinedString);
    }
}
```

Explanation:

- We first import the StringJoiner class from the java.util package.
- Inside the main method, we create a new instance of StringJoiner by providing a delimiter (",") to separate the elements, a prefix ("[") and a suffix ("]") to enclose the final string.
- We then add three string elements to the joiner using the add() method.
- Finally, we call the toString() method of the joiner to get the final joined string, which is stored in the variable joinedString.
- We print the joinedString variable to the console, which outputs the following string: [apple,banana,cucumber].

---

5) **factory design pattern**:-

The Factory Design Pattern is a creational design pattern in Java that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created. This pattern promotes loose coupling between the creator and the product. Here's a basic implementation of the Factory Design Pattern in Java:

```java
// Interface for the product
interface Product {
    void display();
}

// Concrete implementation of Product
class ConcreteProductA implements Product {
    @Override
    public void display() {
        System.out.println("This is Product A");
    }
}

// Another concrete implementation of Product
class ConcreteProductB implements Product {
    @Override
    public void display() {
        System.out.println("This is Product B");
    }
}

// Factory class responsible for creating products
class ProductFactory {
    // Factory method to create products
    public static Product createProduct(String type) {
        if (type.equals("A")) {
            return new ConcreteProductA();
        } else if (type.equals("B")) {
            return new ConcreteProductB();
        }
        return null;
    }
```

```
}

public class Main {
   public static void main(String[] args) {
      // Creating product A
      Product productA = ProductFactory.createProduct("A");
      if (productA != null) {
         productA.display(); // Output: This is Product A
      }

      // Creating product B
      Product productB = ProductFactory.createProduct("B");
      if (productB != null) {
         productB.display(); // Output: This is Product B
      }
   }
}
```

In this example:

- `Product` is the interface for the products created by the factory.
- `ConcreteProductA` and `ConcreteProductB` are concrete implementations of `Product`.
- `ProductFactory` is the factory class responsible for creating instances of `Product`. It contains a factory method `createProduct()` which takes a parameter to determine which type of product to create.
- In the `main` method, we demonstrate how to use the factory to create instances of `Product` without directly instantiating them.

This way, if you need to introduce new products or modify the creation process, you can do so without modifying the client code, promoting flexibility and maintainability.

---

6) **class containing static class inside a static class**

Inside ApplicationProperties.java :-

```
// some annotations are here
class ApplicationProperties{
@Getter
   @Setter
   @Validated
   public static class Bulk {
      private ProjectQuery projectQuery;
      private GrantedProjectQuery grantedProjectQuery;

      @Getter
      @Setter
      public static class ProjectQuery {
         @Min(2)
         private int saveChunkSize;
         private Duration saveDurationDelay;
      }

      @Getter
      @Setter
      public static class GrantedProjectQuery {
```

```
        @Min(2)
        private int deleteChunkSize;
        private Duration deleteDurationDelay;
        @Min(2)
        private int saveChunkSize;
        private Duration saveDurationDelay;
      }
    }
}
```

How to access the properties inside these static classes in a different class?
-> In that class, declare above class
private final ApplicationProperties applicationProperties;

Wherever you need, make a call like following:
applicationProperties.getBulk().getProjectQuery().getSaveChunkSize()

---

7) **Duration** class in java

```
 Duration duration = Duration.ofMinutes(5); // Represents a duration of 5 minutes

      // Retrieving duration values
      System.out.println("Duration in minutes: " + duration.toMinutes()); // Output: 5
      System.out.println("Duration in seconds: " + duration.toSeconds()); // Output: 300

      // Adding duration to a specific time
      java.time.LocalTime currentTime = java.time.LocalTime.now();
      java.time.LocalTime newTime = currentTime.plus(duration); // Adds the duration to the current time
      System.out.println("New time after adding duration: " + newTime);
// Output: New time after adding duration: 12:05:25.123456789
```

---

8) **Thread.sleep(..)** in java

```
 public static void sleepQuietly(Duration duration) {

        Thread.sleep(duration.toMillis());
}
```

---

9) **Instant** class in Java

the **Instant Class** is used to represent the specific time instant on the current timeline.

```
Instant instant = Instant.now();

 // Parsing a string sequence to Instant
Instant inst1 = Instant.parse("2021-02-09T11:19:42.12Z");
```

---

10)

StandardCharsets is a class in Java that provides constants for different character encodings (like UTF-8, ISO-8859-1, etc.) These encodings specify how characters are represented as bytes in a computer's memory.

---

11)
import **com.nimbusds.jose.shaded.gson**.JsonElement;

JsonElement- It can represent different parts of a JSON structure, such as objects and arrays.
methods:-
toString()
isJsonObject()
isJsonArray()

JsonParser.parseString(String str)

```
private static void maskCustomerName(JsonObject jsonObject) {
      if (jsonObject.has("customer")) {
         JsonObject customerJsonObject = jsonObject.getAsJsonObject("customer");
         if (customerJsonObject.has("name")) {
            customerJsonObject.add("name", new JsonPrimitive("****"));
         }
      }
   }
```

---

12)

```
List<String> list = List.of("element1", "element2", "element3");
```

This list is unmodifiable.
---
13) TODOs: Inner class, static class, abstract class, serialization, strong typing, object typecasting, abstract static class

non-static inner class/ regular inner class,
static nested class,
local inner class,
anonymous inner class
---

14)

Coding standard-

1. import statement of java standard library should be at the top. Then comes third party libraries like spring.
But, static imports come before non-static imports.

2. after writing methodName put a space and then write the curly bracket {
---

15) java **regex**- regular expressions
Example I got on internet-

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

//1. define the regex pattern
Pattern pattern = Pattern.compile("w3schools", Pattern.CASE_INSENSITIVE);
```

```
//2.
Matcher matcher = pattern.matcher("Visit W3Schools!");

//3. Matcher with input string
   boolean matchFound = matcher.find();

   if(matchFound) {
    System.out.println("Match found");
   } else {
    System.out.println("Match not found");
   }
---------------------------------

   Pattern pattern = Pattern.compile("name is ([A-Za-z ]+)\\. I am ([0-9]+) years old\\.");

   Matcher matcher = pattern.matcher(input);

   if (matcher.find()) {

    String fullName = matcher.group(1);
    String age = matcher.group(2);

    System.out.println("Full Name: " + fullName);
    System.out.println("Age: " + age);
   }
```

We use the group() function of the matcher object to extract the matched substrings if a match is found. group(1) returns the substring that corresponds to the first capturing group (the name), and group(2) returns the substring that corresponds to the second capturing group (the age).

Static nested class
Static helper method
Dependency injection into static nested class

The Vavr library for Java provides io.vavr.Tuple2 as an immutable way to store two elements of potentially different types, offering a functional approach to handling pairs of data.

```
Tuple2<String, Integer> java8 = Tuple.of("Java", 8);

Tuple3<String, Integer, Double> java8 = Tuple.of("Java", 8, 1.8);
String element1 = java8._1;
int element2 = java8._2();
double element3 = java8._3();
```

Office java training_NotesPart1

Wednesday, January 15, 2025 11:44 AM

Trainer- Jai Prakash Singh
from session 1 to 10

1) Session 1- 15th Jan 2025

Cross-assembler- can generate code for another machine
javac compiler: java code to byte code - .class file
java interpreter: byte code file to output

Java- Write Once Run Anywhere

Oak- former name of java
James Gosling developed it
Sun Microsystems bought by Oracle

Standalone applications	J2SE
Web application	J2EE
Enterprise application	J2EE
Mobile application	J2ME

println- overridden method
---
2) Session 2- 17th Jan 2025

JVM- loads classes
JRE- runtime environment
JDK- tools to compile and debug

identifier- programmer given name
- cannot start with digit
- can start with $ or _

int _num;
int $num;

java is case sensitive language

Access modifiers- public, protected, private, default
default- package private
(e.g. If a method has default access modifier, then it cannot be accessed from outside of the package. Even if you try to import the class and call that method, it won't happen)

protected- (access within the same package + access within subclasses)

built-in packages- io, sql

class name starts with upper case
method name starts with lower case
package name in lower case

- **There are 8 primitives data type in java : -**
- Byte - 8 bits - 1 byte
- Short - 16 bit
- Int - 32 bit
- Long - 64 bit
- Float - 32 bit
- Double - 64 bit
- Char - 16bit unicode character - ASCII - a - 95
- Boolean - 1 bit (true/false)

Instance variable- declared inside class, but outside method
- separate copy for each object, should be non-static

Instance variable will get the default value if not assigned

In the static block, non-static variables cannot be accessed directly.
So, you'll have to create the object and then access them.

You **cannot declare** new static variables inside static block, but you can initialize static variables in a static block

Values of instance variables will be specific to the object.
They will change as the object changes.
---
3) Session 3- 20th January 2025

Local variable- declared inside any method
- must be initialized before use
- only accessible within the method

When instance variable is tried to be accessed from static block like main method- Error- **Non-static variables cannot be referenced from static context**.

I HAVE PREPARED THIS TABLE.

X	Inside static block	Inside non-static block
static variable	- cannot be declared - can be initialized/accessed	- cannot be declared - can be initialized/accessed
non-static variable	- local variable: can be declared - local variable: can be initialized/accessed - instance variable: cannot be initialized, can be accessed using object	- local variable: can be declared - local variable: can be initialized/accessed - instance variable: can be initialized/accessed(using this keyword in current instance of class, or using object to access instance variable of another class)

Static variable-
- declared inside class, outside method, prefix as static keyword
- also known as class variable (because it belongs to class, not the member of class)
- has default value
- called from static as well as non-static context
- single copy for all objects, persistent

final variable-
- final keyword
- constant value, cannot be changed
- cannot be inherited

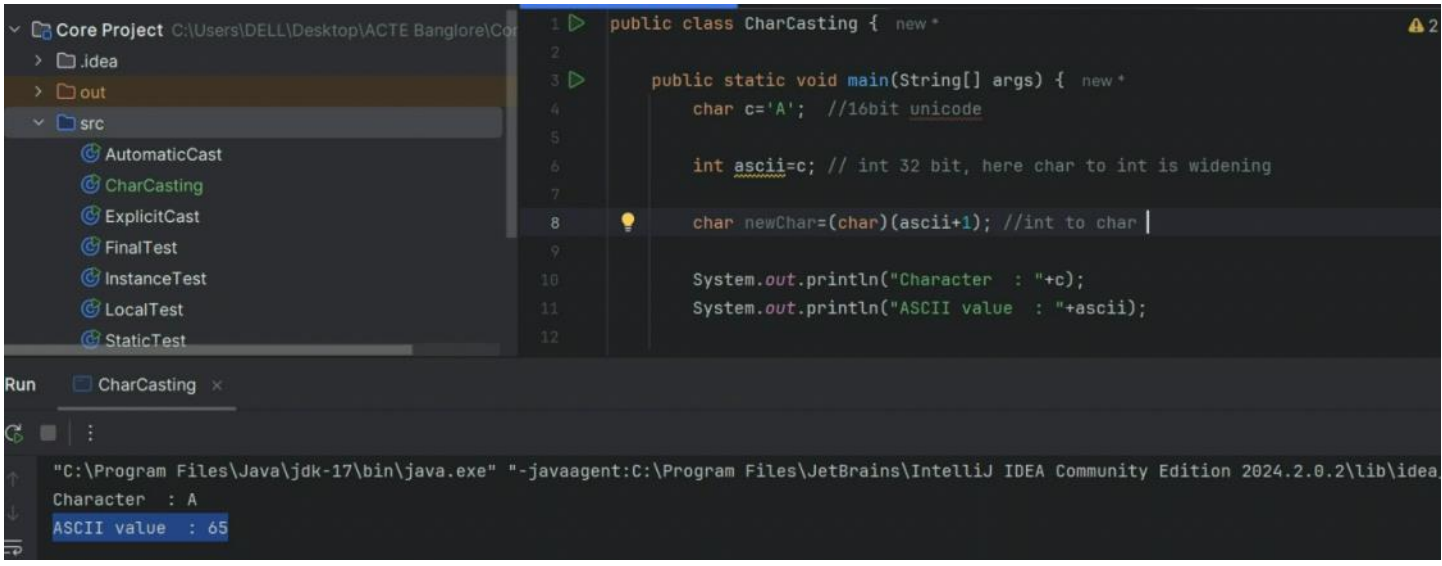
public static final int b = 30;

Typecasting:
1) Automatic casting- Widening (Smaller data type inside larger data type), Upcasting, Implicit casting
2) Manual casting- Narrowing
Data loss may happen in narrowing

Widening:-
Conversion of int to double
int a = 100; //100
double num = a; //100.0

Narrowing:-
double a = 100.5;
int num = (int) a; // 100 <- loss of value
---

4) Session 4- 22nd January 2025



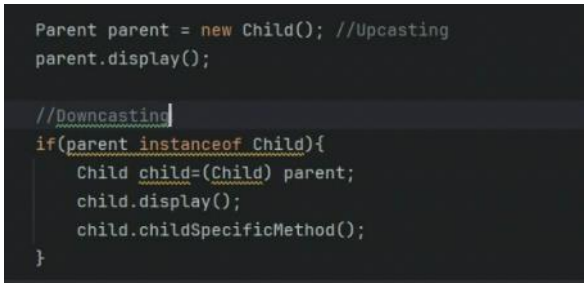
Character casting:-
char c = 'A'
int x = c; //x is 65 - ascii value Widening
char c1 = (char) (x +1); //c1 is B because of ascii value of 66 --Narrowing
---

Upcasting- Parent reference child object
Parent parent = new Child();

Downcasting-
Child child = (Child) parent;

upcasting-
child's method will be called.

In practice, to avoid ClassCastException, it's often wise to use
the **instanceof** operator to check if the reference can be cast to the desired type:



My observation- Parent reference Parent object cannot be converted to Child reference Parent Object.
Parent reference Child object, can be converted to Child reference Child Object

Auto-boxing- primitive data type to wrapper class
//Autoboxing
int x = 10;
Integer integer = x;

Auto-unboxing- wrapper class to primitive data type

//Auto unboxing- automatic conversion
// ~~int a = new Integer(100);~~ //Integer(int)' is deprecated since version 9 and marked for removal

Integer integer2 = 100;
int i = integer2;

Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character

Wrapper classes- useful in Collections in java

// ~~int a = new Integer(100);~~ Integer(int)' is deprecated since version 9 and marked for removal
```
	Integer integer2 = 100;
	int i = integer2;
```

~~int a = new Integer(100);~~ // this conversion is deprecated

---

Coding Standards:

1. Name should reflect the purpose of class
2. Variable name:
Constants in upper case
3. Generally avoid single letter variable declaration.
But single letter variable name can be used in loops.
4. Indentation- Leave one space before {
Open braces should be at the same line as that of declaration.
```
main () {
}
```
5. java doc comment- /** */
6. package name in reverse order- like com.schaeffler ; not ~~schaeffler.com~~

---

5) Session 5- 27th January 2025

Scanner- provide value at runtime
- in java.util package
- it provides methods to read and process input efficiently

System.in - reference of standard input stream
System.out- reference of standard output stream

next(), nextLine(), nextInt()

sc.close()

```
int x = sc.nextInt();
sc.nextLine();
String st = sc.nextLine();
```

Read from String:-

```
String str = "India is my country";
Scanner sc = new Scanner(str);
while(sc.hasNext()){
	System.out.println(sc.next());
}
```

Read from file:-

```
 try {
		File file = new File("src/Lecture1to10/info.txt");
		Scanner sc_new = new Scanner(file);
  // Unhandled exception: java.io.FileNotFoundException
		while(sc_new.hasNextLine()){
		   String line = sc_new.nextLine();
		   System.out.println(line);
		}
		sc_new.close();
	}
	catch(FileNotFoundException fileNotFoundException){
		fileNotFoundException.printStackTrace();
	}
```

---

6) Session 6- 29th January 2025

Control statements- if else , switch
curly braces inside switch case when multiple statements are to be printed.

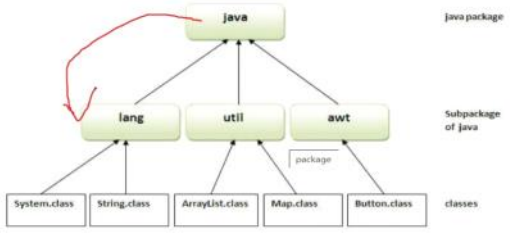
Flow statements- for, while, do-while, for-each loop

_____

7) Session 7- 31st January 2025

i) package in java- namespace that groups related classes and interfaces together
package- should be first statement of program

Steps:
1. Write package statement in .java file (Demo.java)
2. In command prompt- be in same path as that of java file
javac -d . Demo.java
3. Package will get created with respective name as in your program and this package will have the .class file
4. To run this program->
java com.acte.Demo



Using fully qualified name to access the class-

```
package com.acte.custom;
public class StuDriver{
	public static void main(String []args){
		com.acte.edu.Student stu=new com.acte.edu.Student();  //using fully qualified name
		stu.welcome();
	}
}
```

---

private instance variable is not accessible outside the class.
default- package private- not accessible outside the package
protected- access within the package and outside the package through inheritance

_____

8) Session 8- 3rd February 2025

default constructor, parameterized constructor, copy constructor

```
public class User{
	User(){   // no need to write public if there is no inheritance
	}
}
```

In class User, if you write return type like-
public void User(){   // same method name as class name
}
then, this is not an error. It is the warning.

Constructor Overloading:-
- same name, different arguments

private constructor can be created.
- **A private constructor is accessible only within the class itself.**
- **A private constructor restricts object creation and prevents inheritance.**
- **If a class needs to be inherited, its constructor should not be private.**

_____

9) Session 9- 7th February 2025

OOP

1) Encapsulation:-
private instance variables and using getter and setter

2) Inheritance- inheriting the properties,
extends keyword, implements keyword,
code reusability

class being inherited- base class, parent class, super class
class that inherits- subclass, child class, derived class

3) Polymorphism-
Compile time polymorphism- method overloading
Run time polymorphism- method overriding, dynamic binding

4) Abstraction- hiding implementation details and showing only essential features.
Achieved through abstract class or interface.

In interface:-
public static final- variables
abstract - methods by default

We cannot create object of abstract class.

---

10) Session 10- 10th February 2025

Types of inheritance-
1. Single inheritance- Parent class Child extends Parent
2. Multilevel inheritance- class A class B extends A class C extends B
3. Hierarchical inheritance- one parent class- multiple child classes
4. Multiple inheritance-
5. Hybrid inheritance- combination of multiple and hierarchical inheritance

super keyword- to access immediate parent class instance variable, immediate parent class's method
super()- immediate parent class's constructor

Diamond problem- The problem arises when the inheriting class tries to call a method that's defined in the common ancestor. Because there are two paths to the ancestor (creating a diamond shape in the inheritance diagram), the compiler or runtime environment can be ambiguous about which version of the method to call.

Anonymous object- object without reference variable

super() should be the first statement in the constructor.

- Refer to my code examples in the project

Compile:-
javac HelloWorld.java

Analyze the compiled class:-
javap HelloWorld

Office java training_NotesPart2

Monday, February 10, 2025 4:26 PM

from session 11 to 21

11) Session 11- 12th February 2025

Method signature- method name and parameters

Method overloading- method to be called is decided at compile time based on the method signature

Compiler will have to remember only one name.

The return type is not part of the method signature.

Compile-time error:-

```
public class MyClass {

   public int add(int a, int b) {
      return a + b;
   }

   public double add(int a, int b) { // This will cause a compile-time error
      return a + b;
   }
}
```

Method overriding- method signature should be same in parent and child classes
return type must be same or subtype- covariant return type

Overridden method- in parent class
Overriding method- in child class

Method Overloading vs. Method Overriding

Feature	Method Overloading	Method Overriding
Definition	Multiple methods with the same name but different parameters	Child class redefines a method from the parent class
Where it occurs?	Within the same class	Between parent and child classes
Method Signature	Different parameter list	Must be the same as the parent class
Return Type	Can be different	Must be the same or a covariant type
Access Modifiers	No restriction	Cannot reduce visibility (e.g., protected → private is not allowed)
Binding	Compile-time (static binding)	Run-time (dynamic binding)
Performance	Faster (determined at compile time)	Slower (determined at runtime using virtual table)

abstract class

-
- can have constructor
- has abstract method
- can have normal or concrete method
- class that extends the abstract class has to override the abstract methods in abstract class; otherwise that class will also be abstract

```
abstract class Shape{
   protected String shapeName;
   protected int shapeArea;

   public abstract void drawShape();
}

class Circle extends Shape{
   public void drawShape(){
      System.out.println("Circle drawn");
   }
}
```

My learnings:-

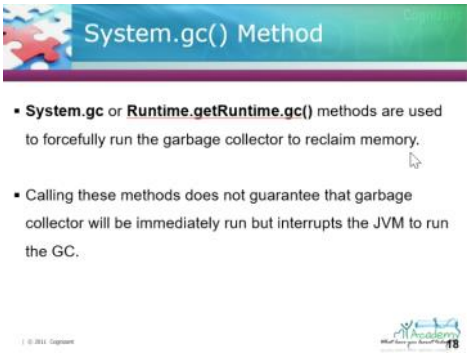
- this is not allowed in static context
- Top-level class cannot have protected as access modifier, but nested class can have it.

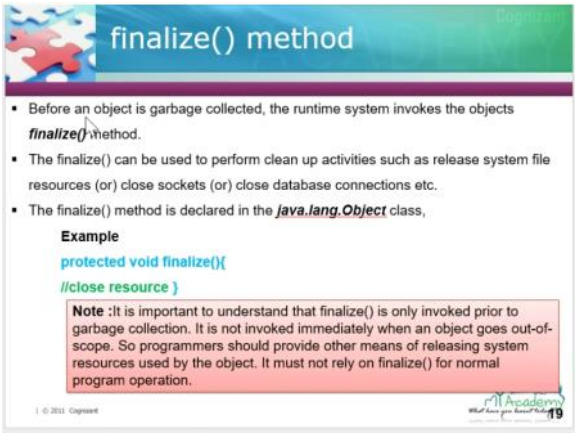
_____

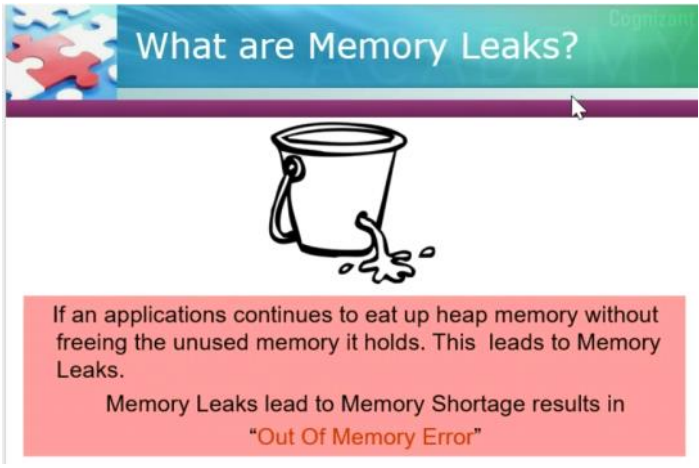
12) Session 12- 14th February 2025

interface- blueprint of class
- contract that implementing classes must follow
- methods- public and abstract
- variables are public static final
- we achieve 100% abstraction
- we can achieve multiple inheritance

Garbage collection- done by JVM
- unused object will be called by JVM
- helps in performance optimization








```
Runtime runtime = Runtime.getRuntime();
        System.out.println("Total memory: "+ runtime.totalMemory());
        System.out.println("Free memory: "+ runtime.freeMemory());

        for(int i=0; i<100; i++){
           new Object();
        }

        System.out.println("Before gc() Free memory: "+ runtime.freeMemory());

        System.gc();

        System.out.println("After gc() Free memory: "+ runtime.freeMemory());
```

_____-

13) Session 13- 19th Feb. 2025

Marker interface- no methods or fields. It's essentially an empty interface.
Its purpose is to "mark" or tag classes that implement it with a specific characteristic or attribute.

Serialization- i) save the state of an object
ii) send objects over a network
iii) Storing serialized objects in a cache can improve performance

Serializable interface is Marker interface.

Interfaces were the standard way to tag classes.

4. **Used in Frameworks & Libraries →** Many Java frameworks (like **Serializable, Cloneable, Remote**) rely on marker interfaces.

Examples of Built-in Marker Interfaces in Java

Marker Interface	Purpose
Serializable	Marks a class for Java object serialization.
Cloneable	Marks a class to allow object cloning (Object.clone()).
Remote	Used in RMI (Remote Method Invocation) for remote objects.
SingleThreadModel (Deprecated)	Used in Servlets to indicate thread safety.

Annotations started from Java 5

Serialization- object to byte stream
- to persist the state of object

ObjectInputstream
ObjectOutputStream

FileOutputStream extends ObjectOutputStream

```
Employee.java   SerializationDemo.java
public class SerializationDemo {  new *
    public static void main(String[] args) {  new *
    }catch (IOException e){
        e.printStackTrace();

    try(ObjectInputStream in=new ObjectInputStream(new FileInputStream( name: "employee.ser"
        Employee deserializedEmp=(Employee) in.readObject();
        System.out.println("Object Deserialized : "+deserializedEmp);
    }catch (IOException | ClassNotFoundException ex){
        ex.printStackTrace();
    }
```

Findings from discussions with senior colleagues:

The data in .ser file here will be in one line only.
If you modify it, then it will be corrupt and deserialization won't happen.

Although Serializable interface has nothing inside it, there must be a check somewhere where Employee class is instanceOf Serializable would be getting checked. So, class Employee2 implements Serializable

Problematic situation of serialization:
class A{ B b;}
class B{ A a;}
Serialization will be problematic because there is recursion

formats for serialization- xml, json, bjson

---

transient- to make the field not persistent

private **transient** String password;
// This field will not be serialized

ecureEmployee.java TransientDemo.java

```
package day13;

import java.io.*;

public class TransientDemo {
    public static void main(String[] args) {
        SecureEmployee emp =new SecureEmployee( name: "Jay", age: 35, password: "mypass123");

        try(ObjectOutputStream out=new ObjectOutputStream(new FileOutputStream( name: "sec
            out.writeObject(emp);
            System.out.println("Serialzed : "+emp);
        }catch (IOException e){
            e.printStackTrace();
        }

        try(ObjectInputStream in=new ObjectInputStream(new FileInputStream( name: "secure_e
            SecureEmployee deserializedEmp=(SecureEmployee) in.readObject();
            System.out.println("Deserialized : "+deserializedEmp);
        }catch (IOException | ClassNotFoundException ex){
            ex.printStackTrace();
        }
```

---

14) Session 14- 21st February 2025
Exception handling

Exception - abnormal condition - interrupts execution flow
Program terminates if it occurs

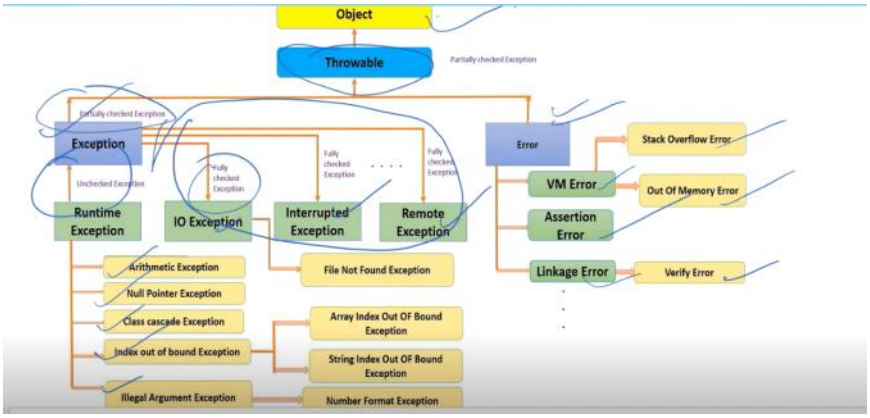
End user input mistakes, network connection problems

Every Exception is a predefined class present in different package.
java.lang.ArithmeticException ----------> **java.lang**
java.io.IOException ----------> **java.io**
java.sql.SQLException ----------> **java.sql**
javax.servlet.ServletException ----------> **javax.servlet**

1. Checked exception- compile-time exception- IOException, ClassNotFoundException
- handled with try, catch

2. Unchecked exception- runtime exception- NullPointer, ArrayIndexOutOfBounds
- Occurs during the execution of program
  - default exception handler- JVM
3. Error- OutOfMemoryError

Throwable- parent class of all Exception classes



Exception Handling Technique : -
In java, there are five types of keyword, which is using in exception handling.

Try - define code that may throw an exception
Catch - handle the exception
Throw - Manually throw an exception
Throws - declare exceptions that a method can throw
Finally - execute always(even if an exception occurs or not)

Rules:

1. **Try must be with one catch block or finally block**
2. **Finally is optional but it execute always**
3. **We can use more than one catch block with one try block**
4. **We catch first the child exception than parent exception**
5. **In case of user defined exception we catch user defined exception first than child exception than parent exception.**

throw- used to throw pre-defined and custom exceptions
But recommended to use 'throw' in custom exceptions.

throws- method level or constructor level

```
public class ExcepTest5 {  new *
    public static void main(String[] args) throws Exception{  new *
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);
        System.out.println("Enter your name : ");
        String name=br.readLine();
        System.out.println("Your name : "+name);
    }
}
```

---

15) Session 15- 24th February 2025

Priority:
1. User defined exception
2. Child class exception
3. Parent class exception

No statement in between try and catch

Methods to print exception related information:
toString(), getMessage(), printStackTrace()

toString()- exception with message
java.lang.ArithmeticException: / by zero

getMessage()- only message
/ by zero

printStackTrace()- exception, its message and place where it occurred
java.lang.ArithmeticException: / by zero
at lecture11to20.exceptionHandling.ExceptionInformationPrint.m2(ExceptionInformationPrint.java:11)
at lecture11to20.exceptionHandling.ExceptionInformationPrint.m1(ExceptionInformationPrint.java:6)
at lecture11to20.exceptionHandling.ExceptionInformationPrint.main(ExceptionInformationPrint.java:21)

Custom exception
- created with Exception class or Throwable class
- Extend Exception class(for checked exception) and RuntimeException class(for unchecked exception)
- write constructor- to initialized message and cause

When a method declares that it **throws** an exception, it's essentially saying, "I'm not going to handle this exception myself; I'm passing the responsibility to the calling method."

- The calling method then has two choices:
  - It can either use a try-catch block to handle the exception.
  - It can also declare that it throws the same exception, passing the responsibility further up the call stack.

---

16) Session 16- 26th February 2025

- java.lang package is imported by default.

1. **String Class -**
   - **String is final class, that means we can't inherit it. We can't create subclass of string class.**
   - **String is an immutable classes, that means we can't change or modify the object of string class**
   - **If we want to change or modify the object of string class in that case new object will be created. The reference of the object will refer to the newly created object.**
   - **Old object will lose their reference, but both the objects will exists in memory. It is a kind of memory wastage.**
   - **We can create object of string class with or without new keyword**
   - **If we create object with new keyword, memory will create in heap.**
   - **Object creation of String without new keyword, memory will created in SCP(String Pool)**

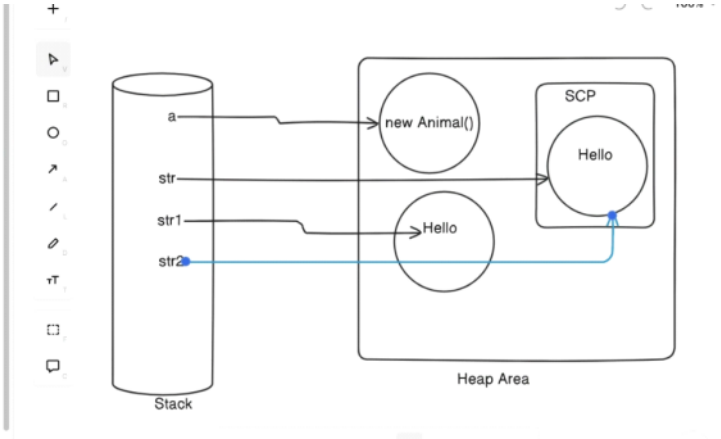
Methods of string class : -
toUpperCase()
toLowerCase()
length()
concate()

String s1= "Hello"; //string pool
String s2= "Hello"; // refers to same string pool

String constant pool (SCP)- part of Heap memory
String literals are stored in common pool and reused

Untitled File

```
Animal a=new Animal();

String str="Hello";
String str1=new String("Hello");
String str2="Hello";
```




```
// intern() method
String a = new String("Java").intern();
String b = "Java";
System.out.println(a==b); //true
```

```
public static void main(String[] args) {
    String str = "Hello";
    String str2="Java";
    str=str.concat( str: "sir");
    System.out.println(str);
    System.out.println(str.toUpperCase());
    System.out.println(str.toLowerCase());
    System.out.println(str.charAt(5));
    System.out.println(str.indexOf('l'));
    System.out.println(str.lastIndexOf( ch: 'l'));
```

```
public class ObjectCount {
  static int objectCount=0;

  public ObjectCount(){
    objectCount++;
  }

  public static void main(String[] args) {
    ObjectCount obj1 = new ObjectCount();
    ObjectCount obj2 = new ObjectCount();
    ObjectCount obj3 = new ObjectCount();
    System.out.println("Number of objects: "+objectCount); //3
```

```
    }
}
```

---

17) Session 17- 3rd March 2025

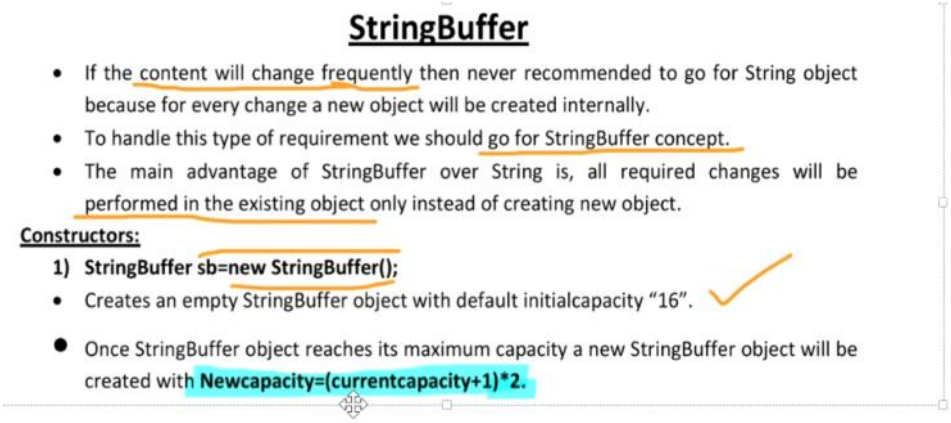
StringBuffer class- mutable class
- final class and we can't extend it. java.lang package
- All the methods are thread-safe

methods:-
reverse(), append()

delete(start, end)

String's method-
concat()

```
StringBuffer sb2 = new StringBuffer(10);
System.out.println(sb2.capacity());
sb2.append("ertyuiopqsd"); // 11 characters
System.out.println(sb2.capacity()); // 10+1= 11 -> 11*2 = 22
```



StringBuffer

- If the content will change frequently then never recommended to go for String object because for every change a new object will be created internally.
- To handle this type of requirement we should go for StringBuffer concept.
- The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object.

Constructors:

1) **StringBuffer sb=new StringBuffer();**
- Creates an empty StringBuffer object with default initialcapacity "16".
- Once StringBuffer object reaches its maximum capacity a new StringBuffer object will be created with **Newcapacity=(currentcapacity+1)*2.**

insert()

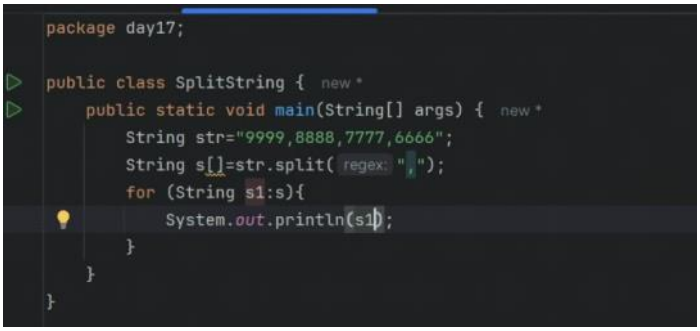
Q. Can we create immutable class in java?

R. Yes, we can create immutable class in java. Any immutable class must follow the following properties :

i. Declare the fields final and private.
ii. Declare the class with final
iii. Not setter only getter
iv. No methods to modify the state of the object
v. Set the data by using parameterized constructor.

substring(startIndex, endIndex)

StringIndexOutOfBoundsException



```
package day17;

public class SplitString {
    public static void main(String[] args) {
        String str="9999,8888,7777,6666";
        String s[]=str.split( regex: ",");
        for (String s1:s){
            System.out.println(s1);
        }
    }
}
```

String tokenizer- break the string into tokens based on delimiters

```java
import java.util.StringTokenizer;

public class StringTokenTest {
    public static void main(String[] args) {
        String country="India USA Uk Russia";
        StringTokenizer obj=new StringTokenizer(country, " ");
        while (obj.hasMoreElements()){
            String token=obj.nextToken();
            System.out.println(token);
        }
    }
}
```

join()

---

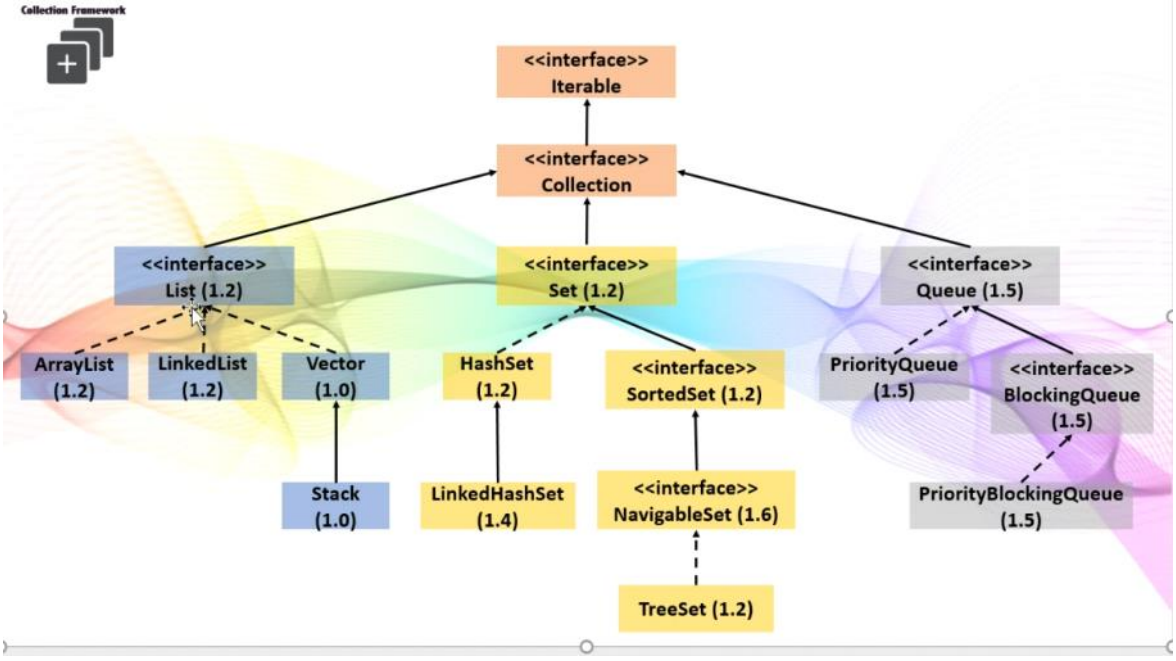
18) Session 18- 5th March 2025

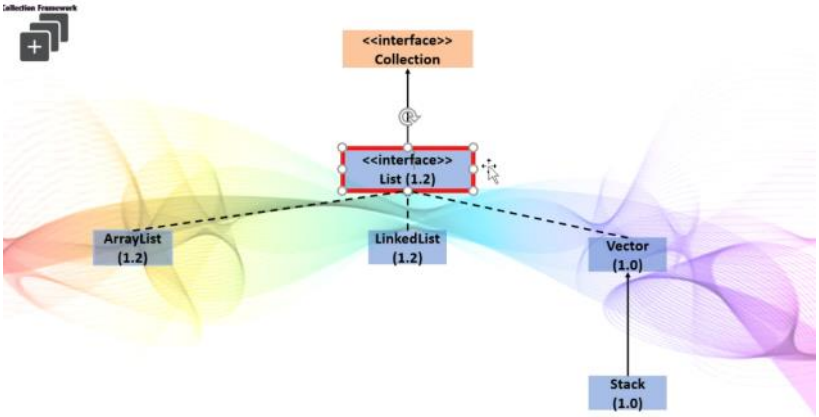
Refer to my code-

Collection framework-

collection framework classes and interfaces in java.util package
Root interface of Collection framework is Collection

Difference between Collection interface and Collections class:
Collections- utility class that contains methods to perform operations





Array List

- Initial Capacity and Incremental Capacity is applicable.
- If the total number of elements exceeds than its capacity, the incremental capacity of the ArrayList is ((Current Capacity * 3)/2) + 1
- Cannot control the growth. That is the incremental capacity is fixed.
- Involves shift operations. That is manipulation with ArrayList is slow because it internally uses an array.
- If any element is removed from the array, all the bits are shifted in memory.
- An ArrayList class can act as a list only because it implements List only.

Array List

- ArrayList is better for storing and accessing data.
- Consumes less memory.
- Memory is continuous.
- Has three overloaded constructors:
- public ArrayList(int initialCapacity)
- public ArrayList()
- public ArrayList(Collection<? extends E> c)

Map is a part of Queue interface

1. **The collection framework classes introduced versions**
   Different classes are introduced in different versions.
2. **Heterogeneous data allowed or not allowed**
   All most all collection framework classes allowed heterogeneous data except two classes
   1. TreeSet 2. TreemMap
3. **Null insertion is possible or not possible.**
   Some classes are allowed null insertion but some classes are not.
4. **Duplicate object are allowed or not allowed**
   Some classes are allowed null insertion but some classes are not allowed

Iterator allows removal of elements during iteration using remove() method.

---

19) Session 19- 7th March 2025

A) LinkedList-

B) Vector- synchronized methods- thread-safe
- dynamic resizing
- indexed access
- implements List interface

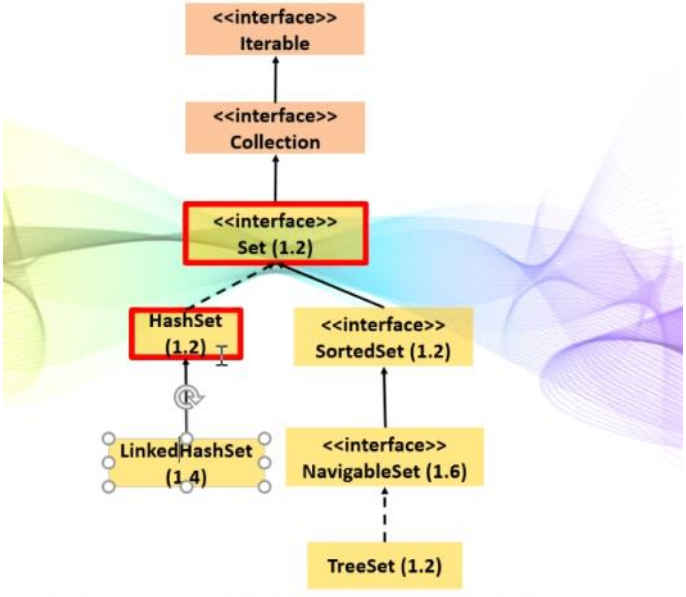
```
Vector<Integer> vector = new Vector<>();

    // Adding elements
    vector.add(10);
    vector.add(20);
    vector.add(30);

vector.get(1) //20
```

Arraylist- multi-threaded
Vector- single-threaded

C) Stack

search() - returns -1 if not found

D) Set



peek()- to return the element on the top of the stack

HashSet

- elements are inserted on the basis of their hash code.

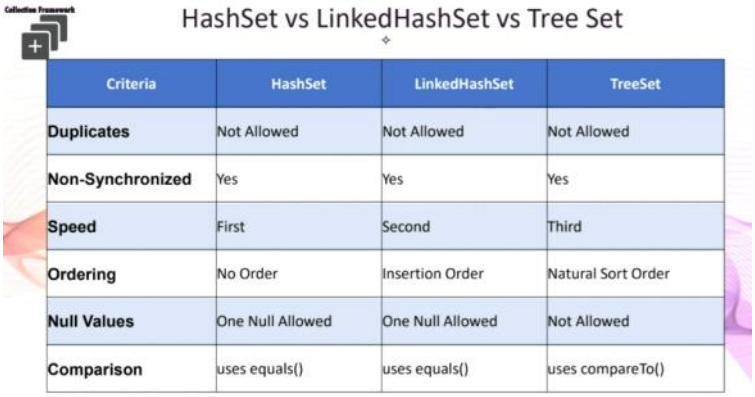
A hash function to map keys to indices in an array (the "hash table").

---

20) Session 20- 10th March 2025

LinkedHashSet- Duplicates are not allowed and insertion order is preserved.
- LinkedList and HashSet

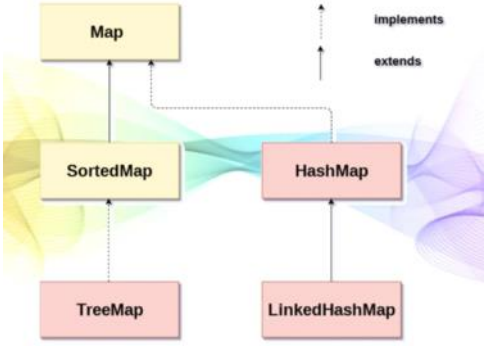
TreeSet
- Heterogenous data is not allowed
- implements SortedSet interface
- objects are stored in a sorted and ascending order



HashSet vs LinkedHashSet vs Tree Set

Criteria	HashSet	LinkedHashSet	TreeSet
Duplicates	Not Allowed	Not Allowed	Not Allowed
Non-Synchronized	Yes	Yes	Yes
Speed	First	Second	Third
Ordering	No Order	Insertion Order	Natural Sort Order
Null Values	One Null Allowed	One Null Allowed	Not Allowed
Comparison	uses equals()	uses equals()	uses compareTo()

Map
- to search, update, or delete elements on the basis of key
- key-value pair = entry
- cannot be traversed. So, convert into set



HashMap- may have one null key and multiple null values
- underlying data structure is HashTable
- heterogeneous value allowed

---

21) Session 21- 12th March 2025

Hash table

Comparable interface- java.lang package
Set interface implements Comparable interface by default

Comparator interface

My notes-

TREESET

- The TreeSet starts at the root of its underlying binary search tree.
- It compares the new element with the root element.

- If the new element is "less than" the root, it moves to the left subtree.
- If the new element is "greater than" the root, it moves to the right subtree.
- This process repeats recursively, comparing the new element with the elements in the subtrees, until it finds the appropriate position to insert the new element.

Comparable vs Comparator interfaces Article by freecodecamp->
Comparable vs Comparator Interfaces in Java – Which Should You Use and When?

Office java training_NotesPart3

Wednesday, April 2, 2025 10:14 AM

from session 22 to 31

22) Session 22- 2nd April 2025

enum

- public enum **DaysOFWeeks**{SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THRUSDAY,FRIDAY,SATURDAY}

Every enum constant represent object of type enum.

Internal Implementation : -

- Class DaysOfWeeks{
  - public static final DayOfWeeks SUNDAY=new DaysOfWeeks();
  - public static final DayOfWeeks MODAY=new DaysOfWeeks();
  - public static final DayOfWeeks TUESDAY=new DaysOfWeeks();
  - public static final DayOfWeeks WEDNESDAY=new DaysOfWeeks();
  - public static final DayOfWeeks THRUSDAY=new DaysOfWeeks();
  - public static final DayOfWeeks FRIDAY=new DaysOfWeeks();
  - public static final DayOfWeeks SATURDAY=new DaysOfWeeks();

}

enum constants are by default public static and final

Compiler generates .class file for enum

Inside switch case we can use enum.

```
package day22;

enum OrderStatus{  2 usages  new *
    PENDING, SHIPPED, DELIVERED, CANCELLED;  1 usage

    public boolean isCompleted(){  no usages  new *
        return this == DELIVERED || this == CANCELLED;
    }
}

class Order{  no usages  new *
    private int orderId;  1 usage
    private OrderStatus status; //enum  1 usage

    public Order(int orderId){  no usages  new *
        this.orderId=orderId;
        this.status=OrderStatus.PENDING;
    }
}
```

Java Date and Time API

- used for logging, scheduling, report generation, financial calculations

1. Legacy API- java.util.Date, java.util.Calendar - not recommended for new applications
   - java.util.Date- mutable and outdated
     - year starts from 1900

mutable- not thread-safe

2. Modern java Time API- java.Time package

---

23) Session 23- 4th April 2025

File folder = new File (path);
- folder.exists()
- mkdir()
- mkdirs()

import java.io.File;

File
- createNewFile()
- getName()

apache pdfbox- for pdf
apache POI- for word, excel, powerpoint

```
package day23;
import java.io.File;
public class CreateFolderExample {
   public static void main(String[] args) {
      File folder=new File("C:/Users/DELL/Desktop/ACTE
Banglore/DemoFolder/SwaraliProject");
// Here both folders- DemoFolder and SwaraliProject were not existing. mkdirs() will create both
folders
      if(!folder.exists()){
         if(folder.mkdirs()){
            System.out.println("Folder Created Successfully....");
         }else {
            System.out.println("Fail to create folder.....");
         }
      }else {
         System.out.println("Folder already exists.......");
      }
   }

}
```

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

      File file = new File("C:/Users/DELL/Desktop/ACTE Banglore/sample.txt");

      try{
         if(file.createNewFile()){
            System.out.println("File created: "+file.getName());
         }else {
            System.out.println("File already exists..");
         }
      }catch (IOException e){
         System.out.println("An error occured.");
         e.printStackTrace();
      }
   }
}
```

---

23) Session 23- Multithreading

Thread based multitasking

Junit uses threads to run test cases in parallel.

Which is better process based multitasking or thread based multitasking?
Thread based multitasking is better. Multitasking threads require less overhead than process based multitasking. Process are heavyweight tasks that require own separate address spaces. Threads, on the other hand are lighter weight. They share the same address space and cooperatively share the same heavyweight process. Inter-process communication is expensive and limited. Context switching from one process to another is also costly.

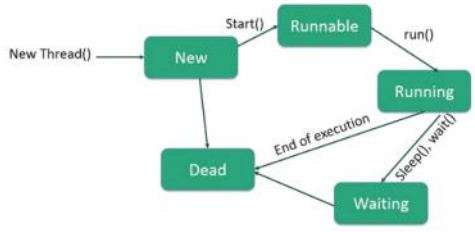
Thread- light-weight process within a process

All java programs have at least one thread known as main thread.

Life Cycle of a Thread (Thread States) : -

A thread can be in one of the five states in the thread. According to sun, there is only 4 states **new, runnable, non-runnable and terminated.** There is no running state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread is controlled by JVM. The thread states are as follows : -

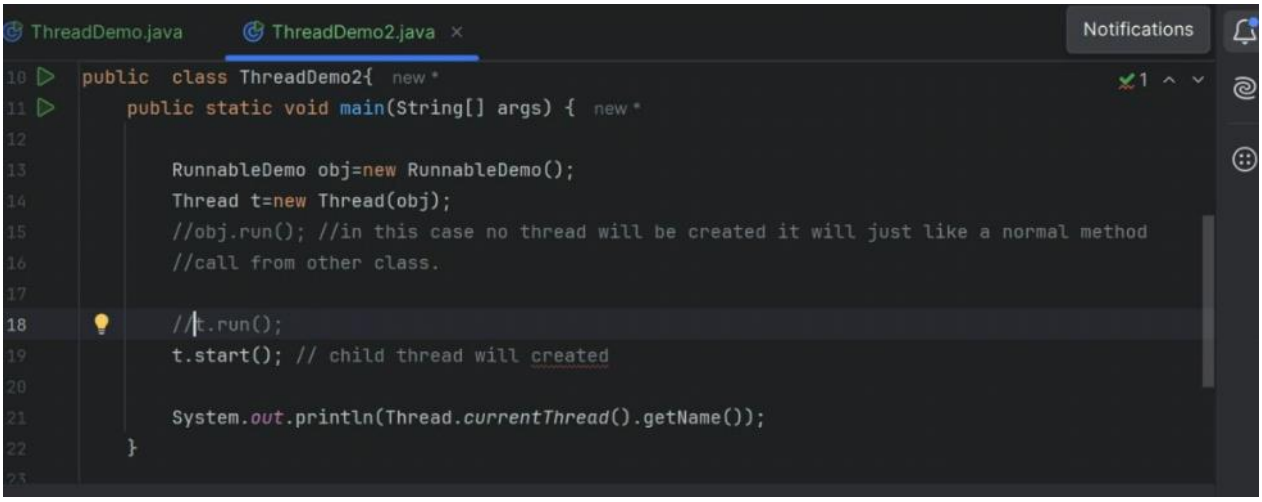
i. New
ii. Runnable
iii. Running
iv. Non-Runnable(Blocked)
v. Terminated



user thread, daemon thread
Garbage collection- done by JVM

Thread scheduler- will decide- which thread will get first chance

Thread.currentThread().getName()



Thread priority-
Minimum- 1, Normal-5, Maximum-10

Priority range- from 0 to 10

Thread.currentThread().getPriority()

setPriority()

yield()
join()

join()- waiting for another thread to finish its execution

The yield() method is a weak hint to the thread scheduler that the current thread is willing to give up the CPU. Its behavior is platform-dependent and not guaranteed.

---

24) Session 24- Thread synchronization

Synchronization- ensures that only one thread can modify one resource at a time

synchronized - on method
OR synchronized on block

```
synchronized(className.class){
      for(int i=1; i<=10; i++){

      }
}
```

wait()

Object Locking : -

Synchronization is achieved using locks. When a thread enters a synchronized method or block, it acquires a lock on the object. Other threads that attempt to enter any synchronized method or block on the same object will be blocked until the lock is released.

notify()- wakes up a single thread waiting on the object monitor
notifyAll()

InterruptedException

synchronized(this)

_____

25) Session 25- Java 8 features

Lambda expression-

if only one parameter- no parenthesis required
We can call the lambda expression just like a method.

```
// normal OOP implementation
      LambdaDemo lambdaDemo = new LambdaDemo();
      lambdaDemo.add(10, 20);

      BiConsumer<Integer, Integer> biConsumer = (a, b)-> System.out.println(a+b);
      biConsumer.accept(10, 20);
      // BiConsumer does not return

      Function<Integer, Integer> function = i-> i*i;
      System.out.println(function.apply(4));
      // Function- returns

      List<String> names = Arrays.asList("Swarali", "Pallavi");
      names.forEach(name-> System.out.println(name));
```

Functional interface- has only one abstract method but it can have multiple default or static methods

@FunctionalInterface- optional annotation
- this annotation will make sure that there is only one abstract method

---

```
@FunctionalInterface
interface FInterface{
   void add(int a, int b);
   default void printName(){
      System.out.println("Welcome to functional interface");
   }
}

public class FunctionalInterfaceDemo {
   public static void main(String[] args) {
      FInterface fInterface = (a, b)-> System.out.println(a+b);
      fInterface.printName();
   }
}
```

Method reference

_____

26) Session 26- time API, Stream, StringJoiner, Parallel sort, Optional

```
LocalDate today = LocalDate.now();
System.out.println(today);

LocalDateTime now = LocalDateTime.now();
System.out.println(now);

DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
System.out.println(today.format(dateTimeFormatter));
```

Stream- we can use them to process elements inside the collection, perform bulk operations on them

```
ArrayList<Integer> al = new ArrayList<>();
al.add(20);
al.add(37);
al.add(99);
al.add(64);

List<Integer> l1= al.stream().filter(i-> i%2==0).collect(Collectors.toList());
System.out.println(l1);

List<String> names= Arrays.asList("Swarali", "Prerana", "Mrunalini");

List<String> filteredNames = names.stream()
        .filter(name -> name.startsWith("S"))
        .collect(Collectors.toList());
filteredNames.forEach(System.out::println);
```

StringJoiner

```
StringJoiner stringJoiner = new StringJoiner(", ", "[", "]");
stringJoiner.add("Swarali");
stringJoiner.add("Prerana");
stringJoiner.add("Pallavi");

System.out.println(stringJoiner);
```

Parallel Sort

Optional

Links

Saturday, May 6, 2023 10:47 PM

Youtube Playlist- Java + DSA + Interview Preparation Course

Notes-
kunal-kushwaha/DSA-Bootcamp-Java: This repository consists of the code samples, assignments, and notes for the Java Data Structures & Algorithms bootcamp of Community Classroom. (github.com)

Link for Kunal's lecture notes and assignments from Java DSA playlist

In my Notes, I'm writing only what's new for me.

Suggestions in the meetings

Thursday, September 14, 2023 8:37 PM

1. String singleTypeQuery= "project.id={0} AND type= ' '{1}' ' ";
   We can use placeholder in the string and replace the value later.
   String query = **MessageFormat.format**(singleTypeQuery, projectId, "Task");

2. Instead of converting CodebeamerTrackerItem class's attributes to TrackerItem class's attributes in Service class, do the conversion in TrackerItem class's constructor itself.
   So, in Service class you can write- map(TrackerItem::new)

   ```
   List<CodebeamerTrackerItem>codebeamerTrackerItems=response.items();
   List<TrackerItem> result= codebeamerTrackerItems.stream().map(TrackerItem::new).collect(Collectors.toList());

   public TrackerItem(CodebeamerTrackerItem codebeamerTrackerItem){
   this(codebeamerTrackerItem.id(),
   codebeamerTrackerItem.name(),
   codebeamerTrackerItem.endDate(),
   TrackerType.fromCBtoALMServiceTracker(codebeamerTrackerItem.typeName()),
   codebeamerTrackerItem.priority().name(),
   codebeamerTrackerItem.status().name()
   );
   }
   ```

3)
   ```
   return Arrays.stream(projectRoles.roles())
   .map(FieldModel::name)
   .map(role->StringUtils.trimWhitespace(role))
   .anyMatch(role->(StringUtils.hasLength(role)));
   ```

Java coding guidelines

Monday, August 28, 2023 11:56 AM

1. Oracle Java coding conventions- https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html
2. Google Java style guide- https://google.github.io/styleguide/javaguide.html

record classes in Java

Monday, July 31, 2023 11:21 AM

Some classes used for data carrying, storage.
Usually we write,
class, its fields, getter-setter methods, toString overridden, equals, hashCode
Alien a1= new Alien(1, "MarsAlien");
Alien a2= new Alien(1, "MarsAlien");

On writing a1.equals(a2)-> false (because they're different objects)
To check the contents, we'll have to override equals() and write our implementation.

Instead, use **record classes**-
record className(dataType1 fieldName1, dataType2 fieldName2, ….) { }
//state variables inside
//state variables are by default private and final

record Alien(int id, String name)

Without writing constructor I can write this.
Alien a1=new Alien(1, "MarsAlien");
Alien a2= new Alien(1, "MarsAlien");
a1.equals(a2) -> we'll get true without overriding equals() because of record classes

Now if I use, System.out.println(a1); , I won't get memory address and won't have to override toString()

default constructor is not present by default. But we can write it.
If we write the default constructor->
Error-> A non-canonical constructor must start with an explicit invocation to a constructor

```
record Employee(int id, String name){
      public Employee(){
        this(0, "");
}
```

All state variables in record are by default final

public record …..

```
record Employee(int id, String name){
      public Employee{    //instead of writing parameters in parameterized constructor
            if(id==0)
               throw new IllegalArgumentException("id cannot be zero");
        }
}
```

```
record Employee(int id, String name){
        static int x= 10;
       //Error- Because this is instance variable- int y= 11
}
public class Test{
       public static void main(String[] args){
              Employee e1= new Employee(145, "swarali");
              System.out.println(e1.id()+ " "+ e1.name() );
       }
}
```

Time complexity- Java data structures

Friday, May 17, 2024 1:20 PM

JAVA COLLECTIONS

Cheat Sheet

List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	O(1)	O(n)	O(1)	O(n)	O(1)	Array
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	Linked List
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)	O(1)	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	O(1)	O(1)	O(1)	O(h/n)	O(1)	Hash Table
LinkedHashSet	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
EnumSet	O(1)	O(1)	O(1)	O(1)	O(1)	Bit Vector
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Redblack tree
CopyOnWriteArraySet	O(n)	O(n)	O(n)	O(1)	O(1)	Array
ConcurrentSkipListSet	O(log n)	O(log n)	O(log n)	O(1)	O(n)	Skip List

Map	Put	Remove	Get	ContainsKey	Next	Data Structure
HashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
LinkedHashMap	O(1)	O(1)	O(1)	O(1)	O(1)	Hash Table + Linked List
IdentityHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Array
WeakHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Table
EnumMap	O(1)	O(1)	O(1)	O(1)	O(1)	Array
TreeMap	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	Redblack tree
ConcurrentHashMap	O(1)	O(1)	O(1)	O(1)	O(h / n)	Hash Tables
ConcurrentSkipListMap	O(log n)	O(log n)	O(log n)	O(log n)	O(1)	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedList	O(1)	O(1)	O(1)	O(1)	O(1)	Array
ArrayDequeue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Array
PrioriritryBlockingQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
SynchronousQueue	O(1)	O(1)	O(1)	O(n)	O(1)	None
DelayQueue	O(log n)	O(1)	O(log n)	O(n)	O(1)	Priority Heap
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(n)	O(1)	Linked List

Java Basics' details

Saturday, May 6, 2023 3:36 PM

Command prompt
javac fileName.java -> to get fileName.class file
java fileName -> to execute

main method is static. Because we want it to run without creating an object of the class in which it is.
main method is supposed to run as soon as program starts, so there cannot be any object creation for it to execute. So, we don't to create the object of class. So, it is static.

---

args is command line argument

```
public static void main(String[] args){
    System.out.println(args[0]);
}
// the output will be 30
```

javac Main.java
java Main 30

System.out.println(args[1])

javac Main.java
java Main 30 "Swarali"

// the output will be Swarali
// If you don't pass the argument, you'll get ArrayIndexOutOfBoundsException

---

package com.Swarali means Swarali is a folder in com folder.

System.out.println()
System class is in java.lang package. It is final class. So, it can't be instantiated.
System has a variable called **out** which is of type PrintStream. **println** method is in PrintStream.

System.out.println(35); // we passed an integer
there's a method written which takes int println(int t) {---}

standard input stream- input from keyboard

Scanner sc= new Scanner(System.in);
we're asking the instance of Scanner to get input from keyboard when sc asks for something
If you'll be taking input from a file, you can provide file as an argument to Scanner.

Primitive data types
- we cannot break them further (e.g. int x= 5; cannot be broken ; but String s="hello"; can be broken to characters)

String is not primitive data type.

Integer x= 30;
Integer is a wrapper class. It makes methods available to you which you can apply on that class's instance.

identifier
literal

int a= 34_000_000;
System.out.println(a); //34000000 underscores are ignored. 34,000,00 instead of comma, we can use underscore_

678687686.789876765 -> float's conversion: **6.7868768E8**

---
float y= sc.nextInt(); // no error
float can store int by default.

int cannot store float. You'll manually have to typecast float to int and then int can store it

System.out.println("नमस्ते");

Unicode

int d= 'न' ;
System.out.println(d); // I got 2344

float * boolean -> float

Maximum value that byte can store is 256

Narrowing type conversion-
int x = 257;
byte y = (int) x;   // 1
257 is beyond the range of byte even after typecasting
y will store x % maxCapacityOfbyte i.e. 257 % 256 = 1

Widening type conversion

float x= 4.34f;
double y= f;

// error- possible lossy conversion from double to float-   float f2= 3/4.0;

float f2= 3/4.0**f** ;

System.out.println(3/4); // 0
---

int x;
System.out.println(x); // error- variable x might not have been initialized

int[] array= new int[4];
System.out.println(array[0]); // 0

// error-
//int y= 5.673;

// not an error-
int y= 5673/1000;

----

final variable needs value to be assigned

Function/methods in java

Saturday, May 13, 2023 12:32 AM

Youtube- Kunal Kushwaha Java DSA playlist

functions in class - called as method
DRY- Don't Repeat Yourself

```
public int getSum(){
        int a= 20;
        int b=30;
        return a+b;

        // Error- unreachable statement (because this is after return statement)
        System.out.println("sum");
}

        int a=10;
        int b=20;
        System.out.println("sum="+a+b);  // sum=1020
        System.out.println(a+b); //  30
```

---

primitives - int, char, short, byte, etc. - pass by value
objects- pass the value of reference variable

```
 public static void main(String[] args) {
      String str= "Swarali";
      String str2= "Swarali";
      String str3= new String("Swarali");
      System.out.println(str==str2); //true
      System.out.println(str==str3); //false

     takeString(str); // it didn't change
     System.out.println(str);   // "Swarali"
   }
```

```
public static void takeString(String s){
        s="Hello World!";
        // here you created new object
    }
```

```
 import java.util.Arrays;
int[] arr= {10, 34, 23};
System.out.println(Arrays.toString(arr));  //  [10, 34, 23]
```

```
public static void main(String[] args) {
      int[] arr= {10, 34, 23};
      modifyArray(arr);
```

```
public static void modifyArray(int[] arrNew){
      arrNew[0]= 32;
    }
```

```
    System.out.println(Arrays.toString(arr));
    // [32, 34, 23]
  }
```

As we are passing the object, the changes are reflected.

```
for(int x=1; x<=4; x++){
        int i=10; // it will be initialized to 10 every time in loop
        System.out.println(i);  // 10
        i+=1;
        System.out.println(i);  //  11
  }
```

Block scope-

```
public static void main(String[] args) {
    int a = 50;
    int x;
    {
      a = 30;
      int c= 50; //cannot be used outside the block
      // error- int x=10
      // because x is already present in the method
    }
    int c= 1000;
    System.out.println(a); // 30
    System.out.println(c); //1000
  }
```

A variable will be accessible inside the block where it is initialized

Shadowing-

```
public class …..{
static int x = 90;
p.s.v.m.(){
    System.out.println(x); // 90
    int x= 1000;
    System.out.println(x); // 1000
    Higher scope(static int x=90) is shadowed.
}

}
```

Variable Arguments-

```
    printVarArgs(4,76,35);
printMultipleVarArgs("Swarali", 63, 3, 2, 10, 87);
```

```
        printVarArgs();  // no argument for VarArgs is not an error except for method overloading

    // var args should be at the end of all parameters
    private static void printMultipleVarArgs(String string, int i, int j, int ...k) {
        System.out.println(string+" "+ i+" "+j);
        System.out.println(k[0]);
    }

    public static void printVarArgs(int ...v){
        System.out.println(Arrays.toString(v));
    }
```

VarArgs can be empty.

But for method overloading it cannot be empty.

```
public static void printVarArgs(int …v){
        System.out.println(v);
}

public static void printVarArgs(String …v){
        System.out.println(v);
}

// error- printVarArgs();  The method printVarArgs(int[]) is ambiguous for the type Methods
```

Arrays and ArrayList in java

Sunday, June 4, 2023 6:56 PM

left side of = is created at compile time
right side of = is created at run time

Dynamic memory allocation- memory is allocated at runtime

In C, C++- there was continuous memory allocation for array elements(continuous blocks are there)

In Java, array objects are in heap. There are no pointers. Heap objects are not continuous.
Array objects may not be continuous- depends on JVM.
---

arr[i] - element at index I
int[] array1= new int[4]; // object in heap memory of size 4 will be created
here in array1, array elements will be zero.

String array elements will be null when not initialized.

String str= null;

null cannot be assigned to primitive data types.
null is a literal.
By default, the value of reference variable is null.
In String array, every element is of type String and every element is a reference variable.
So, when not initialized, String array elements will be null.

My findings-
// WRONG- int x= null;
But, int x= (Integer) null; and Object x= null; are allowed

Length of array- arrayName.length

System.out.println(Arrays.toString(arrayName));

import java.io.*;

```
import java.util.*;
class HelloWorld {
   public static void main(String[] args) {
      int[] arr= new int[4];
      System.out.println(Arrays.toString(arr));   //  [0, 0, 0, 0]
   }
}
```

Watched the video till 34:29

_____

List.of() creates an immutable list with a fixed size, Arrays.asList() produces a modifiable list backed by an array.

_____

static class in java

Monday, August 7, 2023 11:02 AM

static inner class in java-

```
class Outer{

	static class Inner{
		void display(){
			System.out.println("Inside static inner class");
		}
	}
}

Outer.Inner obj= new Outer.Inner();
obj.display();
```

static inner class cannot access the non-static methods of outer class

Functional Programming in Java

Monday, May 8, 2023 10:55 AM

from Udemy SpringBoot course- Master Java Web services and REST Api with springboot in28minutesofficial

Functional programming- introduced in Java 8.

List.of() which takes any number of elements and constructs a list. The list constructed is immutable.
List<String> strList = List.of("Delhi", "Mumbai", "Kolkata");

Functional approach-

```
private static void print(int num){
      System.out.println(num);
   }
main(){
intList.stream().forEach(App::print);  //intList is name of list. App is classname.  :: is method reference
}
```

OR

```
intList.stream.forEach(System.out::println);
```

```
 private static boolean isEven(int number){
      return number%2==0;
   }
   // printing even numbers in list with functional approach
   private static void printEvenNumbersInListFunctional(List<Integer> intList) {
      intList.stream().filter(App::isEven).forEach(System.out::println);
   }
```

Instead of writing isEven function, we can write lambda expression

```
intList.stream().filter(number -> number%2==0).forEach(System.out::println);
```

Courses containing the word "Spring" -

```
courses.stream().filter(str -> str.contains("Spring")).forEach(System.out::println);
```

Printing squares of even numbers in list-

```
//mapping
intList.stream().filter(n->n%2==0).map(m->m*m).forEach(System.out::println);
```

Number of characters in each course name-

```
courses.stream().map(course->course.length()).forEach(System.out::println);
```

```
courses.stream().map(course->course+" "+course.length()).forEach(System.out::println);
```

Spring 6
Spring Boot 11
API 3
Microservices 13

---

java.lang.NullPointerException
Java's solution to solve problems using null's - Optional

Optional class in java

```
import java.util.List;
import java.util.Optional;
import java.util.function.Predicate;

public class PlayingWithOptional {
    public static void main(String[] args) {
        List<String> fruits = List.of("apple", "banana", "mango");

        Predicate<? super String> predicate = fruit -> fruit.startsWith("c");
        Optional<String> optional = fruits.stream().filter(predicate).findFirst();
        System.out.println(optional);
        System.out.println(optional.isEmpty());
        System.out.println(optional.isPresent());
        System.out.println(optional.get());

    }
}
```

Optional.Empty
true
false
Exception

Even though optional had no value, on printing optional, we got Optional.empty and did not get null pointer exception

Optional<String> optional =fruits.stream().filter(predicate).findFirst();

Optional<String> empty= optional.isEmpty();

Lambda expressions

Friday, August 11, 2023 12:46 PM

```
List<Integer> nums= Arrays.asList(1, 5, 32, 8);

java.util.Function has Consumer interface- accept() method

nums.forEach(n -> System.out.println(n)); //we are providing the implementation of accept method
//here n is the parameter of accept
-----------------------------------------------------------------------------------------
@FunctionalInterface
interface A{
	void show();
}

A obj= new A()
{
	public void show(){
		System.out.println("implemented for obj");
	}
};

obj.show();  //to call show() for obj
```

Lambda expressions

Instead of writing this whole syntax,

```
A obj= () -> System.out.println("Lambda expressions");
obj.show();
```

Stream API

Tuesday, August 8, 2023 11:32 AM

youtube- telusko

Stream API

stream of values
Immutable data
stream() method- return type interface Stream<T>

```
List<Integer>nums=Arrays.asList(45,3,21,90);
Stream<Integer>st=nums.stream();
st.forEach(n->System.out.println(n));
```

stream cannot be reused.
This will generate IllegalStateException - st.forEach(n->System.out.println(n));
because st cannot be reused now
st.forEach(n->System.out.println(n));

```
long count=st.count();
System.out.println(count);
```

We are not making any change in the list.

```
Stream<Integer>sortedData=st.sorted();
sortedData.forEach(n->System.out.println(n));
```

map() is used to perform operations
map() gives a new stream

```
Stream<Integer> mappedData= st.map(n-> 2*n);
```

```
List<Integer>nums=Arrays.asList(45,3,21,90);
Stream<Integer>st=nums.stream();
Stream<Integer>mappedSt=st.map(n->2*n);
mappedSt.forEach(x->System.out.println(x));
```

stream(), sorted(), map() - return a new stream

We can write this in one line as:

```
nums.stream().map(n->2*n).forEach(x->System.out.println(x));
```

```
nums.stream().sorted().map(n-> n*n).forEach(x-> System.out.println(n+" "));
```

Here, we created 3 streams. But we are replacing the streams.

```
nums.stream().filter(n->n%2==1).sorted().map(n->2*n).forEach(x->System.out.print(x+""));
```

In filter() method of Stream interface, filter method's parameter is reference of **Predicate which is functional interface and has a method test() with return type boolean**

```
Predicate<Integer> predi = new Predicate<Integer>() {
    @Override
    public boolean test(Integer n) {
     I return n%2==1;
    }
};
```

```
Predicate<Integer> predi = n ->  n%2==1;
```

this is returning n%2==1;
return keyword is not needed.
To the filter(predi) can be written. Instead, we're writing n->n%2==1 directly.

map method's parameter is reference of interface Function which is a functional interface and has apply method.

int sumOfDoubles = nums.stream().map(n->2*n).**reduce**(0,(a,b)->a+b);
System.out.println(sumOfDoubles);

T result = stream.**reduce(identity, binaryOperator);**
identity: This is the initial value that the reduction starts with.
binaryOperator: This is a function that takes two arguments of the same type as the stream elements and produces a result of the same type. It represents the operation that combines two elements into one.

To learn-
parallelStream() and reduce() method

There is a java class CBProject. It has long id, String name; java class Project has long id, String name, String description; I have an array of type CBProject. from every array element I have to access id and call the method projectDescription(int id) which returns description and then put that description in the object of class type Project. projectDescripton() method is written in class Test which has main method also. I want a list of class type Project which will have id and name as that of CBProject but description from projectDescription(). Write the java code using stream API.

```
CodebeamerProject[] codebeamerProjects = response.getBody();
List<Project> projects = Arrays.stream(codebeamerProjects)
.map(P->{
try{
String description=projectDescription(P.id()).getBody();
return new Project(P.id(),P.name(),description);
}catch(IOException e){
throw new RuntimeException(e);
}
})
.collect(Collectors.toList());
```

return new ResponseEntity<>(projects,HttpStatus.OK);

---

anyMatch will return boolean output
StringUtils is in Spring framework

```
return Arrays.stream(projectRoles.roles())
.map(FieldModel::name)
.map(role->StringUtils.trimWhitespace(role))
.anyMatch(role->(StringUtils.hasLength(role)));
```

:: operator in java

Thursday, August 17, 2023 5:46 PM

Method reference in java-

- to refer to methods by their name without invoking them

Types of Method References:

There are four types of method references, each corresponding to a different situation:

- **Reference to a Static Method:**

```java
ClassName::staticMethodName
```

- **Reference to an Instance Method of a Particular Object:**

```java
instance::instanceMethodName
```

- **Reference to an Instance Method of an Arbitrary Object of a Particular Type:**

```java
ClassName::instanceMethodName
```

- **Reference to a Constructor:**

```java
ClassName::new
```

Especially when you're referring to existing methods that match the functional interface's method signature.

They work well when you're using methods that are already defined and can be matched to functional interface methods.

```
List<String> names = Arrays.asList("John", "Alice", "Bob");
names.sort((s1, s2) -> s1.compareTo(s2));

List<String> names = Arrays.asList("John", "Alice", "Bob");
names.sort(String::compareTo);
```

Optional class in java

Friday, August 11, 2023 5:35 PM

optional class-

```
import java.util.Optional;

public Optional<Integer> findAge(String name) {
   // Simulating a database lookup
   if (nameExistsInDatabase(name)) {
      return Optional.of(getAgeFromDatabase(name));
   } else {
      return Optional.empty(); // Indicate that the result is absent
   }
}

public static void main(String[] args) {
   String name = "Alice";
   Optional<Integer> ageOptional = findAge(name);

   if (ageOptional.isPresent()) {
      Integer age = ageOptional.get();
      System.out.println(name + "'s age is: " + age);
   } else {
      System.out.println(name + " was not found in the database.");
   }
}
```

---

enum in java

Wednesday, August 30, 2023 3:51 PM

enum in java
- group of related constant values
- public, static, final constant
- can also have methods and fields
- can also be used in switch statements

```
public enum TrackerType{

TASK("Tasks"),
USER_STORY("UserStories");

@Getter
private String codebeamerTypeName;

TrackerType(String codebeamerTypeName){
this.codebeamerTypeName = codebemaerTypeName;
}
}
```

The TASK("Tasks") line initializes the appTypeName field of the TASK enum constant with the value "Tasks".
This means that when you use the TASK enum constant, its associated appTypeName field will have the value "Tasks".

Comparing enums->

```
EnumType1 enum1Value = EnumType1.VALUE1;
EnumType2 enum2Value = EnumType2.A;
if (enum1Value == EnumType1.VALUE1) // Compare enum values
```

```
public class DayOfWeek {
   public enum Day {
      MONDAY("Monday"), TUESDAY("Tuesday"), WEDNESDAY("Wednesday"),
      THURSDAY("Thursday"), FRIDAY("Friday"), SATURDAY("Saturday"),
      SUNDAY("Sunday");

      private final String dayName;

      Day(String dayName) {
         this.dayName = dayName;
      }

      public String getDayName() {
         return dayName;
      }
   }
}
```

String dayName has value of MONDAY, TUESDAY, etc.

In Java, the **orElseThrow**() method is a part of the Optional class, which is used to handle situations where a value might be absent.
Essentially, an Optional object can either have a value (be non-empty) or not have a value (be empty).
The orElseThrow() method is used when you want to retrieve the value contained within an Optional if it exists, but throw an exception if the Optional is empty.

```
// Create an empty Optional
Optional<String> optionalWithoutApple = Optional.empty();

// This will throw a RuntimeException if the Optional is empty
String fruit = optionalWithoutApple.orElseThrow(() -> new RuntimeException("No fruit found!"));
```

Method Overloading

Friday, December 15, 2023 5:19 PM

Method overloading-

- Return type is not part of method signature when considering method overloading.
- If 2 methods have same name and parameter list, but differ in their return type, it'll result in a compilation error.
- Error-
  public void printLine(){ }
  public int printLine(){ }

- This is allowed but **this is not overloading**-
  public void printLine(){ }
  public int printLine(String x) { }

JDK, JVM

Saturday, December 16, 2023 4:36 PM

.class file can run on any system with compatible JVM installed.

- platform independent file
- byte code

Java bytecode- set of instructions that are executed by Java Virtual Machine.

Amazon Corretto is distribution of Open JDK.

Write Once Run Anywhere (WORA)

interface methods

Monday, December 18, 2023 9:45 PM

interface has all abstract methods.

** **In Java, starting from version 8, interfaces can have default and static methods that provide a default implementation.** These methods are not abstract, as they have a defined implementation. This was introduced to support backward compatibility for existing codebases that use interfaces.

```
interface MyInterface {
    // Abstract method (implicitly public and abstract)
    void abstractMethod();

    // Default method with a default implementation
    default void defaultMethod() {
        System.out.println("Default implementation of defaultMethod");
    }

    // Static method with a static implementation
    static void staticMethod() {
        System.out.println("Static implementation of staticMethod");
    }
}
```

If you don't write anything before return type of method in interface, it'll be considered as abstract method. So, if you want to implement it, either write "static" or "default" keyword before it.

List

Monday, January 8, 2024 11:10 AM

```
List<String> list1 = new ArrayList<>();
list1.add("One");
list1.add("Two");
list1.add("Three");

List<String> list2 = new ArrayList<>();
list2.addAll(list1);
```

Bugs and solutions

Tuesday, January 23, 2024 6:10 PM

1) Constructor makes call to non-final methods
   - Make the method final.

2) There was only one element in Arrays.asList()

It asked to use Collections.singletonList
When I used List.of(), spotbug was gone.

Project Build

Tuesday, February 20, 2024 3:05 PM

The build process involves resolving dependencies, compiling source code, running tests, and generating the executable JAR file or WAR file. The output of the project build can be deployed to a server or container for execution.

Github Reviewer of particular file in project

Tuesday, February 27, 2024 5:00 PM

How to assign a person as reviewer in github for a particular file of project->

.github folder-

CODEOWNERS

***** @joglesar_SGGIT

After * write the github account name of person who is the code owner of entire project

to make the person reviewer of only particular file in github, write path src/main/resources/templates/questionnaire.json @sonawhis_SGGIT

Extra learning

Tuesday, June 27, 2023 12:13 PM

1) Iterating over List with list inside it-

```
Iterator<Father> i = fatherList.iterator();
while(i.hasNext()) {
   Fatherf=i.next();
   System.out.println("Father: "+ f.getName());
   System.out.println("Children:");
   Iterator<Children> ci = f.getChildrens().iterator();
   while(ci.hasNext()) {
       Childrenc=ci.next();
       System.out.println(c.getName());
   }
}
```

2) There is a big list of type List<TrackerItem> result.
   From result, I want particular items having type as "User Stories" using stream API.

   ```
   List<TrackerItem> userStories=
   result.stream().filter(a->"UserStories".equals(a.type())).collect(Collectors.toList());
   ```

TODOs

Tuesday, May 7, 2024 7:14 PM

Learn-

CollectionModel

objectMapper.writeValueAsString()

Stream<Arguments>

Locale

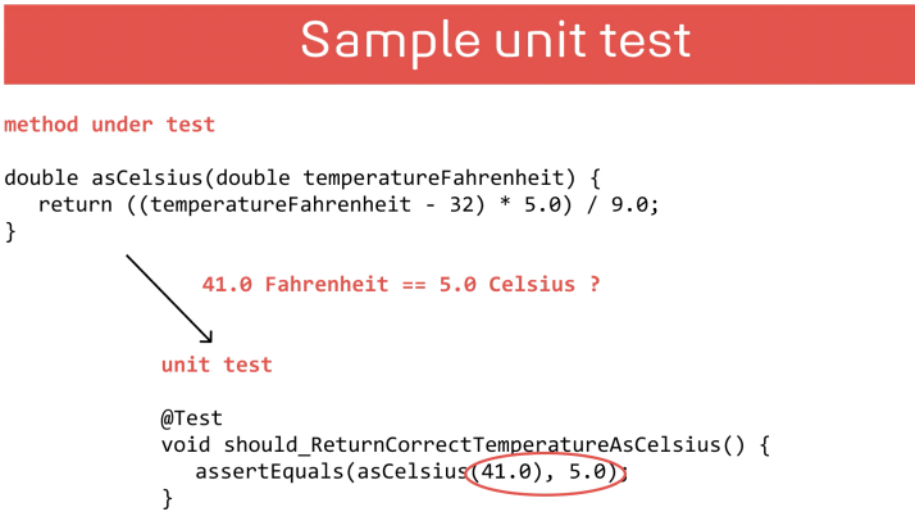
Course- JUnit5 Java Unit Testing

Wednesday, April 10, 2024 12:57 PM

Udemy Course- Learn practical Java Unit Testing with JUnit 5 in just 1 hour

1) Unit testing theory:-

Unit testing- testing single units of software, typically methods in java
If you can't write unit test for your class, then the chances are your class design is wrong.
Unit tests are typically written by software developers and not by software testers.
Unit tests should be more than functional and UI tests.
Junit- standard framework for java, often combined with Mockito
JUnit 4 released in 2006
JUnit 5 released in 2017



---

2)

In this course- the code is written in HealthyCoder project. (I've imported this project in my laptop and going to write tests here)

I have to write the tests only.
Test name should be classNameTest

```
@Test
void test(){
fail("Not yet implemented");
}
```

assertTrue()

```java
package com.healthycoderapp;

import static org.junit.jupiter.api.Assertions.*;

class BMICalculatorTest {

    @Test
    void test() {
        assertTrue(BMICalculator.isDietRecommended(89.0, 1.72));
    }

}
```

---

3)

A project created in eclipse- .zip file-> to be imported in intelliJ-> Process:

File->New-> Project from existing sources-> Import project from external model-> eclipse-> few times click next-> select jdk-> Ok

Marking the test folder- test sources root:
src-> test-> right click-> mark directory as-> test sources root

By marking the directory as a test sources root, IntelliJ can then appropriately configure the build process to compile and run the tests separately from the application code.

Similarly as above, mark the 'resources' folder in test as 'test resources root'.

How to create new test in IntelliJ?
select class name inside the file-> Ctrl+Shift+T-> Create new test-> Testing library- JUnit5 -> Ok
If you get an error, Add JUnit5 to classpath

How I solved the error- Source release 17 requires target release 17 ?

- It happens when we're using older version of compiler to compile newer version of source code.

File-> project structure-> Project sdk 17,
Modules-> Sources-> Lang. level 17
Modules-> Dependencies-> Module sdk-> project sdk 17-> Check the box Junit 5.8

---

4)
The name of the test should be intuitive.

---

5) **assertTrue, assertFalse, assertThrows, Executable**

```java
@Test
void shold_Return_True_WhenDietRecommended(){   // use intuitive name instead of test()

    //given
    var weight1= 89.0; //kg
    var height1= 1.5;   //meter
    var weight2= 59.0;
    var height2= 1.7;

    //when
    var result1= BMICalculator.isDietRecommended(weight1, height1);
    var result2= BMICalculator.isDietRecommended(weight2, height2);
```

Use **org.junit.jupiter.api**.Assertions;

Use **org.junit.jupiter.api.function.Executable**;

```
@Test
  void should_ThrowArithmeticException_when_HeightZero(){

    // We want to check whether exception is thrown when height is 0.0

    //given
    var weight1= 58.0;
    var height1= 0.0;

    //when

    // boolean result= BMICalculator.isDietRecommended(weight1, height1);
    // The above line will throw exception and there would be no chance to check assertThrows
     // So, use Executable
    Executable executable= ()-> BMICalculator.isDietRecommended(weight1, height1);

    //then
    assertThrows(ArithmeticException.class, executable);

  }
```

---

6)
Need of **assertAll()**

If there're two assertions in the test and if the test is failing at first assertion, then we don't know whether it's failing at first assertion only or for remaining assertions also.

```
//then
//assertEquals(1.82,coderWorstBMI.getHeight()); //expected , actual
//assertEquals(98.0,coderWorstBMI.getWeight());

assertAll(
      ( ) ->assertEquals(1.82,coderWorstBMI.getHeight()),
      ( ) ->assertEquals(98.0,coderWorstBMI.getWeight())
);
```

---

7)
assertNull()

```
@Test
```

```
  void shouldReturnNull_CoderWithWorstBMI_WhenCoderListEmpty(){

    // method findCoderWithWorstBMI() should return null when list is empty

    //given
    List<Coder> coders = new ArrayList<>();

    //when
    Coder coderWorstBMI= BMICalculator.findCoderWithWorstBMI(coders);

    //then
    assertNull(coderWorstBMI);
  }
```

---

8)

assertArrayEquals() - to test array equality

```
@Test
  void shouldReturnCorrectBMIScore_WhenCoderListNotEmpty(){
    //given
    List<Coder> coders= new ArrayList<>();
    coders.add(new Coder(1.80, 60.0));
    coders.add(new Coder(1.82, 98.0));
    coders.add(new Coder(1.82, 64.7));

    double[] expectedBMIScores= {18.52, 29.59, 19.53};

    //when
    double[] actualBMIScores= BMICalculator.getBMIScores(coders);

    //then
    // check expected BMI scores w.r.t. actual BMI scores

   //WRONG:- assertEquals(expectedBMIScores, actualBMIScores);  addresses are compared instead of values

    assertArrayEquals(expectedBMIScores, actualBMIScores);

  }
```

---

9)

If the method to be tested in static, you can call it with class name.
If the method is not static, you need to create an object in the test.

If you want object of the class to be created before every test- @**BeforeEach**

```
class DietPlannerTest {

  private DietPlanner dietPlanner;

// Before each test new DietPlanner instance will be created

  @BeforeEach
  void setup(){
    this.dietPlanner = new DietPlanner(20,30,50);
```

```
}

@Test
void should_ReturnCorrectDietlPlan_When_CorrectCoder(){
    //given
    Coder coder = new Coder(1.82, 75, 26, Gender.MALE);

    DietPlan expectedDietPlan = new DietPlan(2202, 110,73,275);

    //when
    DietPlan actualDietPlan= dietPlanner.calculateDiet(coder);

    //then
    // WRONG:- assertEquals(expectedDietPlan, actualDietPlan);  addresses will be compared

    assertAll(
            ()-> assertEquals(expectedDietPlan.getCalories(), actualDietPlan.getCalories()),
            ()-> assertEquals(expectedDietPlan.getFat(), actualDietPlan.getFat()),
            ()-> assertEquals(expectedDietPlan.getCarbohydrate(), actualDietPlan.getCarbohydrate()),
            ()-> assertEquals(expectedDietPlan.getProtein(), actualDietPlan.getProtein())
    );

}
```

---

10)

@AfterEach - to specify a method that should be executed after each test method execution, whether the test passes or fails. It is typically used to clean up test resources or reset test data after each test case.

---

11)

@BeforeAll

- any name and **static**
- uses- setting up database servers, starting servers
- before all unit tests

@AfterAll

---

12)

@ParameterizedTest

@ValueSource

In the following example, the weight's values specified in @ValueSource will be injected in coderWeight.
Double coderWeight is used because @ValueSource annotation in Junit5 only supports wrapper classes.

```
@ParameterizedTest
@ValueSource(doubles= {70.0, 89.0, 95.0, 110.0})
void should_Return_True_WhenDietRecommended_NewWay(Double coderWeight){

    //given
    var weight = coderWeight;
    var height= 1.72;

    //when
    var result= BMICalculator.isDietRecommended(weight, height);

    //then
    assertTrue(result);
```

---

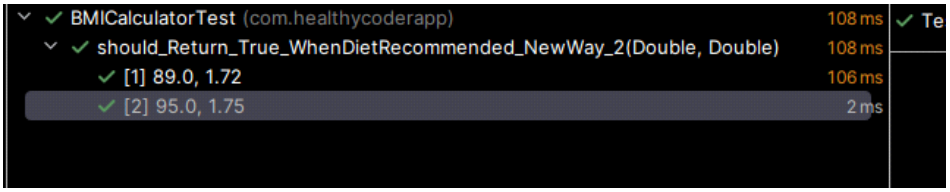
13)
@ParamterizedTest

@**CsvSource** - to provide comma separated values for parameterized test

```
@ParameterizedTest
@CsvSource(value={"89.0,1.72", "95.0 ,1.75"})  //comma separated values- weight, height
void should_Return_True_WhenDietRecommended_NewWay_2(Double coderWeight, Double coderHeight){

    //given
    var weight = coderWeight;
    var height= coderHeight;

    //when
    var result= BMICalculator.isDietRecommended(weight, height);

    //then
    assertTrue(result);
}
```

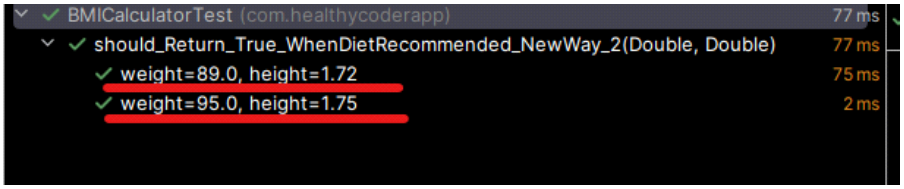
---

14)



Better way of displaying above values:

```
@ParamterizedTest( name = "weight={0}, height={1}")
@CsvSource(value={"89.0,1.72", "95.0 ,1.75"})
void should_Return_True_WhenDietRecommended_NewWay_2(Double coderWeight, Double coderHeight){

.....
}
```



---

15)

```
@ParameterizedTest(name= "weight={0}, height={1}")
@CsvFileSource(resources = "/diet-recommended-input-data.csv", numLinesToSkip = 1)
void should_Return_True_WhenDietRecommended_NewWay_3(Double coderWeight, Double coderHeight){

    //given
    var weight = coderWeight;
    var height= coderHeight;

    //when
    var result= BMICalculator.isDietRecommended(weight, height);

    //then
    assertTrue(result);
  }
```

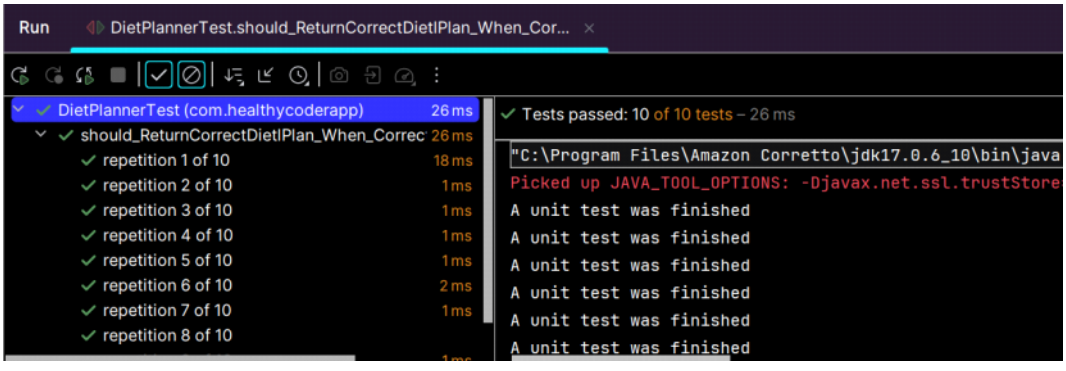
Put the .csv file in src/test/resources. Its first line is weight, height. This line should be skipped by test because we nee d weight and height values. So, use **numLinesToSkip** attribute.

---

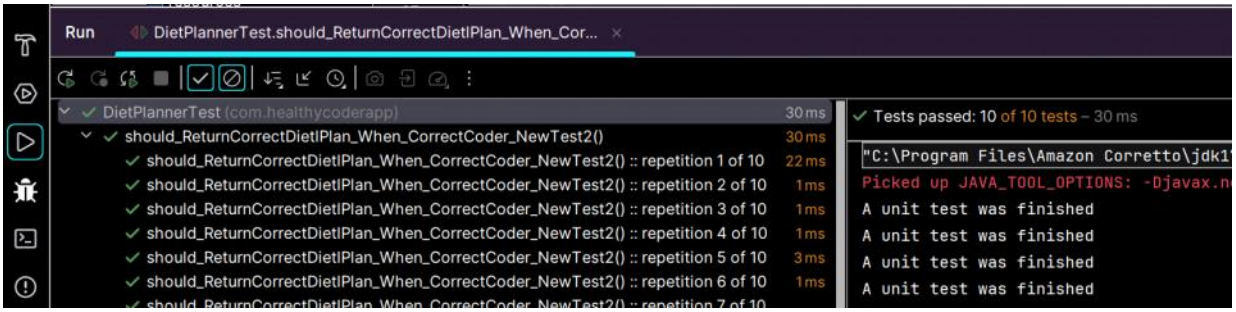
16)

The repetition of test is needed if we're generating random values inside the test and the test needs to run for those values. Don't use repeated test if they don't make sense for your test.

@RepeatedTest(10) //to repeat the test 10 times



@RepeatedTest(**value**= 10, **name**= **RepeatedTest.LONG_DISPLAY_NAME**)
After using long display name, you can see the method name for every repetition.



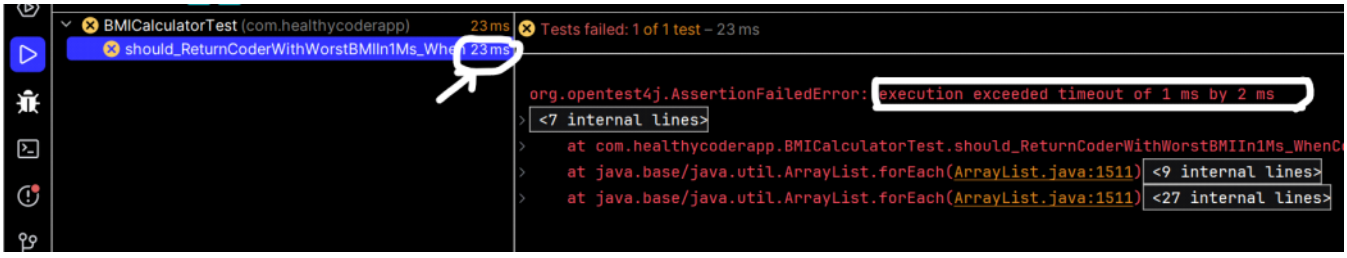
---

17) assertTimeout

Performance unit test with JUnit5

```
@Test
  void should_ReturnCoderWithWorstBMIIn1Ms_WhenCoderListHas1000Elements(){

    //given
    List<Coder> coders= new ArrayList<>();
    for(int i=0; i<1000; i++){
       coders.add(new Coder(1.0+i, 10.0+i));
    }

    //when
    Executable executable = () -> BMICalculator.findCoderWithWorstBMI(coders);

    //then
    assertTimeout(Duration.ofMillis(1), executable); // we expect 1 ms time for executable
     // assertTimeout- only checks the time needed for executable, didn't consider the time to insert 1000 elements

  }
```

assertTimout - only checks execution time of Executable lambda function



23ms time in the above image- is the entire time including inserting 1000 elements

In the right side-> execution exceeded timeout of 1 ms by 2 ms => means execution time of executable was 3 ms

---

18) assumeTrue

performance unit tests might not needed if we're using production environment instead of local machine.

assumeTrue(this.environment.equals("dev")); // Test will be ignored if condition isn't met
This can be written as first line inside the test.
If the variable value doesn't match with dev, the test will be ignored.

```
class …. {
	private String environment= "prod";

}
```

_____

19)

@Nested

We can group the tests inside a class.
Annotate inner class(class inside a class) with **@Nested** and write tests inside it.

_____

20)

@DisplayName(" ") - can be used on inner class or method, name of the test is written inside

@Disabled - to ignore the test

@DisabledOnOs(OS.WINDOWS) - to ignore the test on particular operating system

_____

21)

Tests written in JUnit5 can't work in JUnit4.
Tests written in JUnit4 can work in Junit5 using an annotation.

_____

22)

Test-Driven Development (TDD):- Tests are written before the code is implemented.

- Increased code quality
- Unit tests are ready once you finish your implementation
- Modern and agile approach

_____

23)

HOMEWORK:

It's time for some additional practice! Your task is to import RealEstateApp in a way similar to HealthyCoderApp and write as many unit tests as you can.
RealEstateApp is a project stub for real estate agents. There are three classes available:
1. Apartment (with a given area and price).
2. ApartmentRater, which rates the price/area ratio for a given apartment. There are three possible ratings: 0 (best price/ar ea ratio), 1 or 2 (worst price/area ratio).
3. RandomApartmentGenerator, which generates an apartment with a random price and area. If you invoke the constructor with no parameters, the default values are used: minimum area of 30.0 square meters and minimum price per square meter of 3000.0. You can also sp ecify your own minimum area and price. In either case, the maximum values are: mimum values * 4.0.

Here are my suggestions as to what tests you can write:
1. For ApartmentRater:
a. should_ReturnCorrectRating_When_CorrectApartment -- write a parameterized test with different values
b. should_ReturnErrorValue_When_IncorrectApartment
c. should_CalculateAverageRating_When_CorrectApartmentList
d. should_ThrowExceptionInCalculateAverageRating_When_EmptyApartmentList

2. For RandomApartmentGenerator (since values are randomly generated, repeat the tests multiple times):
a. should_GenerateCorrectApartment_When_DefaultMinAreaMinPrice
b. should_GenerateCorrectApartment_When_CustomMinAreaMinPrice

If you can come up with any other tests, that's even better! The solution to the exercises, however, will only contain the te sts mentioned above.

ProjectLearnings- JUnit5

Friday, May 17, 2024 7:44 PM

1) **Arguments** in JUnit5

The import statement `import org.junit.jupiter.params.provider.Arguments;` is used to include the `Arguments` class from the `org.junit.jupiter.params.provider` package in JUnit 5.

The `Arguments` class is a part of JUnit 5's parameterized tests feature. It is used to create instances of arguments that will be provided to a parameterized test.

In the code you've selected, `Arguments` is used in the `provideTestParameters` method to create a stream of arguments. Each `Arguments` instance represents a set of parameters that will be passed to the `shouldReturnQuestionnaireWhenRequestByProjectIdAndGatewayType` test method.

Here's a simplified example of how `Arguments` might be used:

```java
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

import java.util.stream.Stream;

class ExampleTest {

    static Stream<Arguments> provideTestParameters() {
        return Stream.of(
            Arguments.of("param1", "param2"),
            Arguments.of("param3", "param4")
        );
    }

    @ParameterizedTest
    @MethodSource("provideTestParameters")
```

```
    void testMethod(String param1, String param2) {
        // Test code here
    }
}
```

In this example, `provideTestParameters` returns a `Stream` of `Arguments`. Each `Arguments` instance is created with `Arguments.of()`, and contains the parameters that will be passed to `testMethod`. The `@MethodSource` annotation on `testMethod` specifies that the parameters for this test should be provided by `provideTestParameters`.

---

2) @**MethodSource**

There is a class which has this method-

```
@SuppressWarnings("unused")
    static Stream<Arguments> usersInRoles() {
        // role param is specified for developer purpose to stress what expected role user should have (still role is resolved based on setup data)
        return Stream.of(
            Arguments.of(Constant.PSCR_USERNAME, Role.PSCR),
            Arguments.of(Constant.PSCR_SUBSTITUTE_USERNAME, Role.PSCR_SUBSTITUTE),
            Arguments.of(Constant.PM_USERNAME, Role.PROJECT_MANAGER)
        );
    }
```

Other classes need to use this method in @MethodSource. So, those classes can inherit the class which has the above method.

Course- Mockito: Next Level Java Unit Testing

Friday, May 17, 2024 5:30 PM

Udemy course- Learn the latest Mockito 5 for Java unit testing in just 2 hours and take your developer career to the next level!

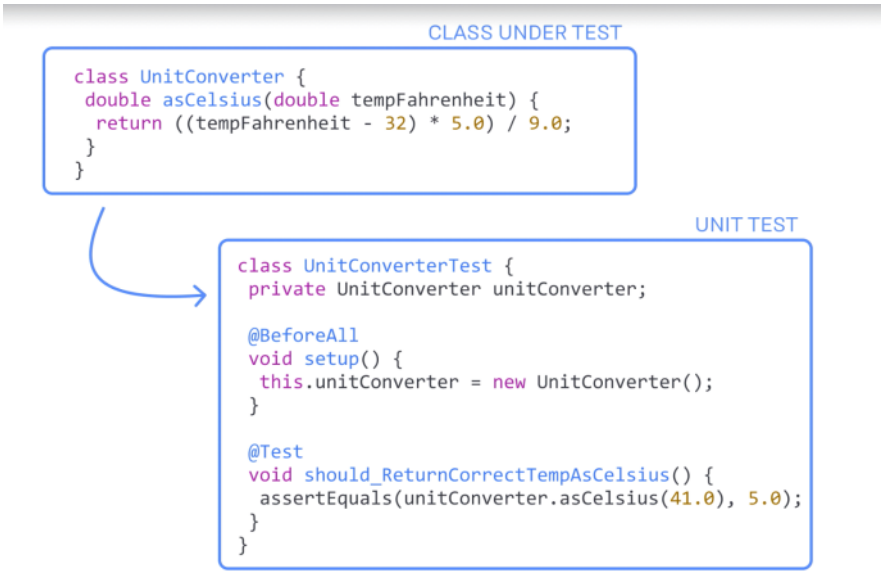
Refer HappyHotelApp project in my Java folder in self-study folder.

Java topics used in this course:

- Collections.singletonList()
- Arrays.asList()
- Collections.emptyList()
- LocalDate class in java

---

1)



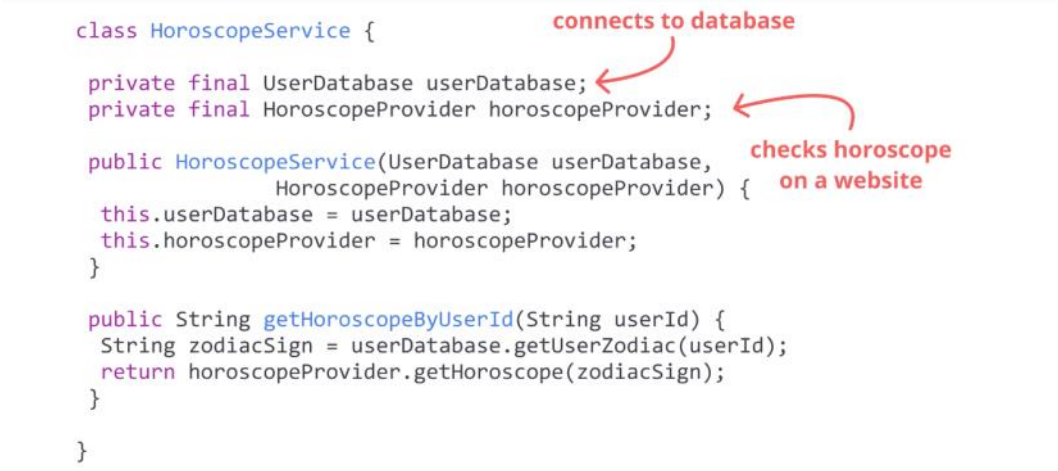
Single responsibility principle- Each java class should do only one thing.

---

2) **Test for class having dependencies, Need of mocking**

Problem Statement:-
Horoscope Service - Get zodiac sign from user id and get horoscope from zodiac sign.

Code:-



Test:-
Inefficient test-

```
public class HoroscopeServiceTest {

 private HoroscopeService horoscopeService;

 @BeforeAll
 void setup() {
  this.horoscopeService = new HoroscopeService(
     new UserDatabase(), new HoroscopeProvider());
 }

 @Test
 ...
}
```

In the above approach it might also happen that HoroscopeServiceTest fails because of problem in its dependencies.
This test class should only fail only if HoroScopeService has bugs.
To write unit tests for HoroscopeService, we should only test its functionality. Ideally we should create userdatabase and horoscopeProvider.

Solution:- **Create mock objects which pretend to be the dependencies**.

For classes having external dependencies, we can write unit tests using mock.

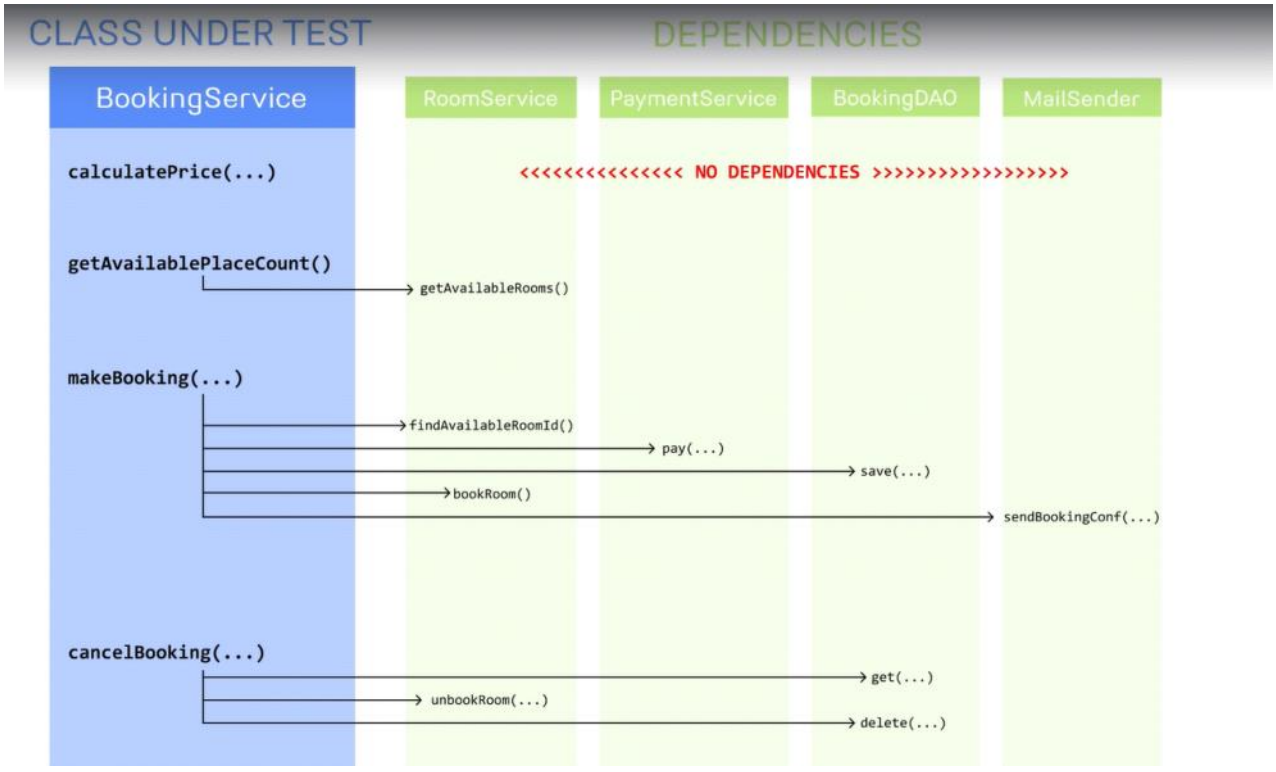
---

3) **i)Mocking frameworks ii)Identifying the dependencies in the methods of class**

Junit does not provide mocking capability.

Mocking frameoworks:-
Mockito, Jmockit, EASYMOCK

Mockito framework is used by most of the Java developers.



---
4)

Mockito 5 requires Java 11 or higher. Core features stay the same in Mockito 3, 4 and 5.

If the following setting is done, project can identify Mockito library.

How I solved the error- Source release 17 requires target release 17 ?

- It happens when we're using older version of compiler to compile newer version of source code.

File-> project structure-> Project sdk 17,
Modules-> Sources-> Lang. level 17
Modules-> Dependencies-> Module sdk-> project sdk 17-> Check the box Junit 5.8
---
5) **mock method**

```
class Test02DefaultReturnValues {

    2 usages
    private BookingService bookingService;

    2 usages
    private PaymentService paymentServiceMock;
    2 usages
    private RoomService roomServiceMock;
    2 usages
    private BookingDAO bookingDAOMock;
    2 usages
    private MailSender mailSenderMock;

    @BeforeEach
    void setup(){
        this.paymentServiceMock= mock(PaymentService.class);
        this.roomServiceMock= mock(RoomService.class);
        this.bookingDAOMock= mock(BookingDAO.class);
        this.mailSenderMock= mock(MailSender.class);

        this.bookingService = new BookingService(paymentServiceMock, roomServiceMock, bookingDAOMock, mailSenderMock);
    }
```

The constructor of class BookingService has parameterized constructor with above dependencies.

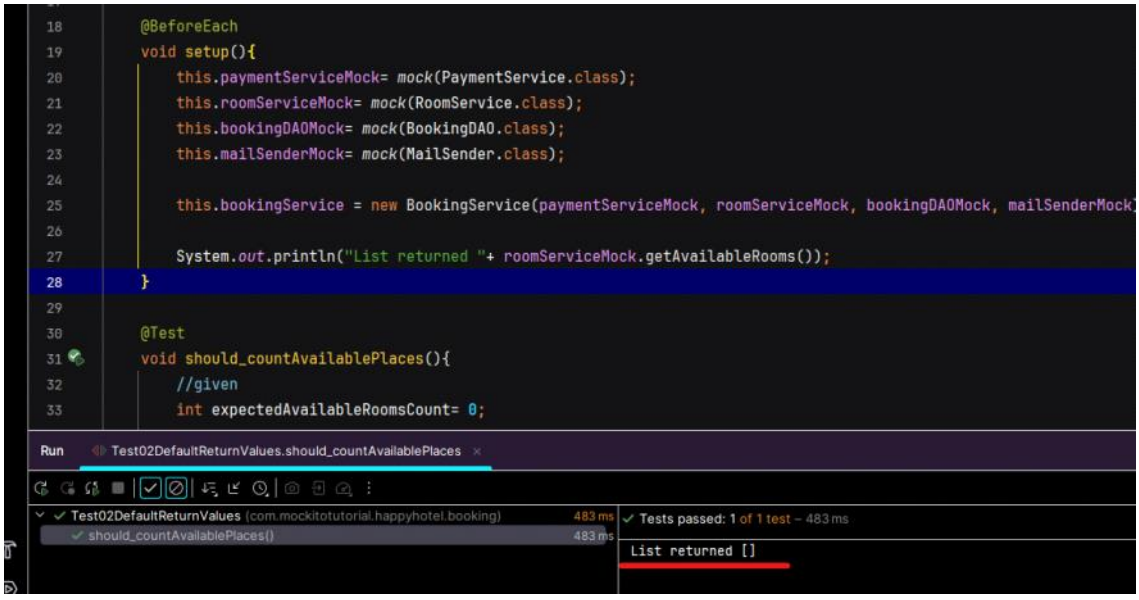
import static org.mockito.Mockito.**mock**;

---

6) **Default values returned by mock**

A method in class BookingService is calling a method in class RoomService.
Although we have mock(RoomService.class), it doesn't contain any data.

The method in RoomService returns a list. But, the mock will return empty list.



---

non-static initialization block in java- The block will execute for every instance of that class.

---

7) **when** , **thenReturn**

```java
class Test03ReturningCustomValues {

    2 usages
    private BookingService bookingService;

    2 usages
    private PaymentService paymentServiceMock;
    3 usages
    private RoomService roomServiceMock;
    2 usages
    private BookingDAO bookingDAOMock;
    2 usages
    private MailSender mailSenderMock;

    @BeforeEach
    void setup(){
        this.paymentServiceMock= mock(PaymentService.class);
        this.roomServiceMock= mock(RoomService.class);
        this.bookingDAOMock= mock(BookingDAO.class);
        this.mailSenderMock= mock(MailSender.class);

        this.bookingService = new BookingService(paymentServiceMock, roomServiceMock, bookingDAOMock, mailSenderMock);
    }

    // getAvailablePlaceCount() method in BookingService is calling getAvailableRooms() method in RoomService
    // So we need data in RoomService mock
    // when getAvailableRooms() of RoomService is called, we need a List<Room>

    @Test
    void should_CountAvailablePlaces_When_OneRoomAvailable() {
        //given
        when(this.roomServiceMock.getAvailableRooms())
                .thenReturn(Collections.singletonList(new Room( id: "Room 1",  capacity: 5)));

        int expected = 5;

        //when
        int actual= this.bookingService.getAvailablePlaceCount();

        //then
        assertEquals(expected, actual);
    }
```

When getAvailablePlaceCount() method in BookiService will be called, it will call getAvailableRooms() method of RoomService. We can use **when and thenReturn** to return the required data for getAvailableRooms() method.

```
List<Room> rooms = Arrays.asList(new Room("Room 1", 5), new Room("Room 2", 3));

    //given
    when(this.roomServiceMock.getAvailableRooms())
        .thenReturn(rooms);
    int expected= 8;
```

---

8) **Chaining of thenReturn**() for check values for multiple calls

```
@Test
  void should_CountAvailablePlaces_when_CalledMultipleTimes() {

    //chaining of thenReturn() to return different values at different calls
```

```
    //given
    when(this.roomServiceMock.getAvailableRooms())
        .thenReturn(Arrays.asList(new Room("Room 1", 4), new Room("Room 2", 3)))
        .thenReturn(Collections.emptyList());

    int expectedFirstCall= 7;
    int expectedSecondCall = 0;

    //when
    int actualFirstCall= this.bookingService.getAvailablePlaceCount();
    int actualSecondCall = this.bookingService.getAvailablePlaceCount();

    //then
    assertAll(
        ()->   assertEquals(expectedFirstCall, actualFirstCall),
        ()-> assertEquals(expectedSecondCall, actualSecondCall)
    );
// Or we can write separate assertEquals() lines

  }
```

---

9) when, **thenThrow**, assertThrows

A method in class BookingService is calling another method in class RoomService which is throwing exception.
We want to check whether this exception is thrown.
thenThrow- to mock the method to throw the given exception

```
 @Test
  void should_ThrowException_when_NoRoomAvailable(){
    //given
    BookingRequest bookingRequest= new BookingRequest("1", LocalDate.of(2020, 01, 01),
        LocalDate.of(2020, 01, 05), 2, false);

    when(this.roomServiceMock.findAvailableRoomId(bookingRequest))
        .thenThrow(BusinessException.class);

    //when
    Executable executable = ()-> bookingService.makeBooking(bookingRequest);

    //then
    assertThrows(BusinessException.class, executable); // expected exception class, executable
  }
```

---

10) **Argument matchers**

any() - used for objects, doesn't work for primitives
anyDouble(), anyBoolean
anyString()- does not match a null String
eq() - to match specifoc non-null argument values

For primitives (e.g. double, integer, boolean), we can't use **any**().

```
when(this.paymentServiceMock.pay(any(),eq(400.0)))
.thenThrow(BusinessException.class);
```

---

11) Verifying behaviour

```
verify(mock).someMethod();
verify(mock, times(10)).someMethod();
verify(mock, atLeastOnce()).someMethod();
```

verifyNoInteractions(paymentServiceMock);

//then
verify(paymentServiceMock,times(0)).pay(any(),anyDouble());

//When prepaid is true**, check whether pay() method is called internally with correct arguments**
verify(paymentServiceMock).pay(bookingRequest,400.0);

//to check no. of times the method is called
verify(paymentServiceMock,times(1)).pay(bookingRequest,400.0);

//pay() method should not be called again.
verifyNoInteractions(paymentServiceMock);

---

12) **spy**

```
class ABC{

private BookingDAO bookingDAOmock;

@BeforeEach()
void setup(){
    this.bookingDAOmock = spy(BookingDAO.class); // Creating a Mockito spy of the BookingDAO class.
}

}
```

mock - dummy object with no real logic
spy- real object with real logic than we can modify

A Mockito **spy** is a partial mock. It means that most calls to the spied object (in this case BookingDAO) are real and not mocked. However, you can still stub certain methods if you want them to behave differently for the purposes of your test.
In Spy we can mock some methods of class, while keeping the real behavior of others.

Mock:- when(mock.method()).thenReturn()

Spy:- **doReturn**().**when**(spy).method()

---
13) Mocking void methods:-

doThrow(new BusinessException()).**when**(mailSenderMock).sendBookingConfirmation(any());

doNothing().**when**(mailSenderMock).sendBookingConfirmation(any());

doNothing() is default behavior of void method

---
14) **ArgumentCaptor, capture()**

- allows you to capture an argument passed to a method in order to inspect it. This can be useful when you want to check that a method was called with the correct parameters.

```
class ABC{

    private ArgumentCaptor<Double> doubleCaptor; // to capture Double type argument

    @BeforeEach
    void setup(){
        this.doubleCaptor = ArgumentCaptor.forClass(Double.class);
    }
```

```
    @Test
      void should_PayCorrectPrice_when_InputOk(){
        //given
        BookingRequest bookingRequest= new BookingRequest("1", LocalDate.of(2020, 01, 01),
            LocalDate.of(2020, 01, 05), 2, true);

        //when
        bookingService.makeBooking(bookingRequest);

        //then
        verify(paymentServiceMock).pay(eq(bookingRequest), doubleCaptor.capture());
        double capturedArgument = doubleCaptor.getValue();

        assertEquals(400.0, capturedArgument);

      }

    @Test
      void should_PayCorrectPrices_when_MultipleCalls(){
        //given
        BookingRequest bookingRequest= new BookingRequest("1", LocalDate.of(2020, 01, 01),
            LocalDate.of(2020, 01, 05), 2, true);

        BookingRequest bookingRequest2= new BookingRequest("1", LocalDate.of(2020, 01, 01),
            LocalDate.of(2020, 01, 02), 2, true);

        List<Double> expectedValues = Arrays.asList(400.0, 100.0); //expected values

        //when
        bookingService.makeBooking(bookingRequest);
        bookingService.makeBooking(bookingRequest2);

        //then
        verify(paymentServiceMock, times(2)).pay(any(), doubleCaptor.capture());

        List<Double> capturedArguments = doubleCaptor.getAllValues();

        assertEquals(expectedValues, capturedArguments);

     // double capturedArgument = doubleCaptor.getValue();
    // assertEquals(400.0, capturedArgument);  WRONG in this case- Only last value will be checked since there are 2 calls to pay()
    method

      }
}
```

---

15) **@ExtendWith(MockitoExtension.class) , @InjectMocks , @Mock , @Spy , @Captor**

- used in JUnit5 to enable Mockito annotations for mocking objects
- tells the JUnit platform to use the MockitoExtension which initializes fields annotated with @Mock, @Spy, @Captor, etc. in your test class.

If you use annotations like @Mock, you won't need to use the methods.

```
@ExtendWith(MockitoExtension.class)
class Test11Annotations {

  @InjectMocks  //will inject mock, spy
  private BookingService bookingService;

  @Mock
  private PaymentService paymentServiceMock;
```

```
@Mock
private RoomService roomServiceMock;

@Spy
private BookingDAO bookingDAOMock;

@Mock
private MailSender mailSenderMock;

@Captor
private ArgumentCaptor<Double> doubleCaptor; // to capture double
```

---

16) **BDD** (Behavior Driven Development), **given**(), **willReturn**(), **then**(), **should**()

when and thenReturn are methods used in Mockito's traditional style of stubbing, while **given** and **willReturn** are used in Mockito's BDD (Behavior Driven Development) style of stubbing.

In BDD, before writing any code, developers, testers, and business stakeholders define the expected behavior of the system in a language that everyone understands. This is often done through examples, which describe how the application should behave in different scenarios. These examples then serve as a guide for developers when they write the code, and for testers when they test the system.

	BDD style
verify(paymentServiceMock, times(1)).pay(bookingRequest, 400.0);	**then**(paymentServiceMock).**should**(**times**(1)).pay(bookingRequest, 400.0);
when(this.roomServiceMock.getAvailableRooms()) .thenReturn(Collections.singletonList(new Room("Room 1", 5)));	**given**(this.roomServiceMock.getAvailableRooms()) .**willReturn**(Collections.singletonList(new Room("Room 1", 5)));

---

17) **Strict stubbing, lenient()**

stubbing- defining the behavior of classes.

If your stubbing is not used- Mockito says -> **Unnecessary stubbing detected.** and test fails **UnnecessaryStubbingException**

To keep unnecessary stubbing in place, Mockito provides **Lenient()** method.

lenient().when(paymentServiceMock.pay(any(),anyDouble())).thenReturn("1");

---

18) **Mocking static methods**

From Mockito 3.4, mocking static methods is possible.

In pom.xml, replace mockito-core by mockito-inline.

Define the behavior of mocked static class.

For the following code to work, I need Java version downgrade to Java 11 or Java 16.

```
@Test
  void should_CalculateCorrectPrice(){

    try(MockedStatic<CurrencyConverter> mockedConverter = mockStatic(CurrencyConverter.class)){

      //given
```

```
        BookingRequest bookingRequest = new BookingRequest("1", LocalDate.of(2020, 01, 01),
              LocalDate.of(2020, 01, 05), 2, false);

        double expected = 400.0 * 0.8;
        mockedConverter.when(()-> CurrencyConverter.toEuro(anyDouble()))
                    .thenAnswer(inv-> (double) inv.getArgument(0)*0.8);

        //argument from toEuro() method is taken and multiplied by 0.8
        //getArgument(0) -> 1st argument passed to toEuro() method

        //when
        double actual = bookingService.calculatePriceEuro(bookingRequest);

        //then
        assertEquals(expected, actual);
    }
```

---

19) **mocking final methods**

mock of final method works for mockito-inline dependency, but not for mockito-core dependency.

From the standpoint of testing, private methods don't exist. Private methods can't be mocked.

Mock the actual non-private method within the class, that calls the private method.

If you want to mock a private method, something is wrong with your class design.

---

20)
Earlier Mockito
New Mockito- mocks for final methods, mocks for static methods, mocking objectconstruction

Powermock
- testing framework like mockito
- Doesn't provide complete support to JUnit 5

---

1. Mockito 5 requires **Java 11 or higher**.
2. Mockito 5 uses **mockito-inline by default**.

You already know mockito-inline from the videos about mocking static, final and private methods.
In Mockito 3/4, you had to change the Maven dependency to mockito-inline. In Mockito 5, you get this by default when using mockito-core.
If you upgrade to Mockito 5 and some of your mocks fail because of mockito-inline, you can still use the old subclass mockmaker for those mocked classes. Mockito introduced a special annotation setting for that.

```
@ExtendWith(MockitoExtension.class)
public class SampleTestClass {

 @Mock(mockMaker = MockMakers.INLINE)
   Person inlineMockPerson;

....
}
```

---

21)
Mockito in SpringBoot project

@MockBean- to replace beans with mocked beans

@SpringBootTest- entire spring context will be initialized

@AutoCOnfigureMockMvc- will use real beans unless we use @MockBean

@SpyBean

TODO-
Test06 file- Write correct code for test

@Mock
@Mock is used to simulate the behavior of real objects in a controlled way. It allows you to create a mock version of a class so you can test specific interactions or behaviors without relying on actual implementations or external resources.

You can then specify how the mock should behave using Mockito methods like when() and thenReturn()

ArgumentCaptor in Mockito

Monday, May 19, 2025 2:39 PM

```
ArgumentCaptor<EmailRequest> emailRequestCaptor = ArgumentCaptor.forClass(EmailRequest.class);
verify(notificationServiceHttpAdapter, times(3)).sendEmail(emailRequestCaptor.capture());

List<EmailRequest> capturedEmailRequests = emailRequestCaptor.getAllValues();
EmailRequest firstEmailRequest = capturedEmailRequests.get(0);
EmailRequest secondEmailRequest = capturedEmailRequests.get(1);
EmailRequest thirdEmailRequest = capturedEmailRequests.get(2);
```

This is written in the then or assert section of the test.

List of Design patterns

Friday, July 25, 2025 2:45 PM

Creational design patterns-

Singleton

Factory

Abstract factory

Builder

Structural design patterns-

Adapter

Decorator

Facade

Composite

Behavioral design patterns-

Observer

Strategy

Command

State

Chain of responsibility

Singleton design pattern

Monday, July 28, 2025 1:03 PM

```
package brandNewJavaLearning.designPatterns;

// Only one government

// Singleton design pattern
class GovernmentOfIndia{
   static GovernmentOfIndia governmentOfIndia = new GovernmentOfIndia();

   // static keyword here-guarantees that only one instance of the class exists
   // and is shared across all calls to getInstance()

   private GovernmentOfIndia(){
   }

   public static GovernmentOfIndia getInstance(){
      return governmentOfIndia;
   }

}

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

      GovernmentOfIndia govt1 = GovernmentOfIndia.getInstance();
      GovernmentOfIndia govt2 = GovernmentOfIndia.getInstance();

      System.out.println(govt1 == govt2);  // true

   }
}
```

Factory design pattern

Monday, July 28, 2025 1:03 PM

```
package brandNewJavaLearning.designPatterns;

// Factory design pattern

class NotificationFactory{

   public Notification getNotificationObject(String notificationType) {
      if (notificationType == null) {
         throw new IllegalArgumentException("Notification type cannot be null");
      }
      switch (notificationType) {
         case "Email":
            return new Email();
         case "SMS":
            return new SMS();
         default:
            throw new IllegalArgumentException("Unknown notification type: " +
notificationType);
      }
   }
}

class Main{
   public static void main(String[] args) {
      NotificationFactory notificationFactory = new NotificationFactory();
      Notification notification = notificationFactory.getNotificationObject("SMS");
      notification.sendNotification();
   }
}

interface Notification{
   public void sendNotification();
}

class Email implements Notification{

   @Override
```

```
    public void sendNotification() {
        System.out.println("Email sent!");
    }
}

class SMS implements Notification{

    @Override
    public void sendNotification() {
        System.out.println("SMS sent!");
    }
}
```

Builder design pattern

Monday, July 28, 2025 1:03 PM

```
class Book {

    private final String name;
    private final String author;
    private final int pages;

    // Private constructor to enforce the use of the Builder
    private Book(Builder builder) {
        this.name = builder.name;
        this.author = builder.author;
        this.pages = builder.pages;
    }

    // Getters for accessing the fields
    public String getName() {
        return name;
    }

    public String getAuthor() {
        return author;
    }

    public int getPages() {
        return pages;
    }

    // Static nested Builder class
    public static class Builder {
        private String name;
        private String author;
        private int pages;

        public Builder setName(String name) {
            this.name = name;
```

```
            return this;
        }

        public Builder setAuthor(String author) {
            this.author = author;
            return this;
        }

        public Builder setPages(int pages) {
            this.pages = pages;
            return this;
        }

        public Book build() {
            return new Book(this);
        }
    }
}

class Main {
    public static void main(String[] args) {
        Book book1 = new Book.Builder()
                .setName("Atomic Habits")
                .setAuthor("James Clear")
                .setPages(300)
                .build();

        Book book2 = new Book.Builder()
                .setName("The Secret Adversary")
                .setPages(350)
                .build();

        System.out.println("Book 1: " + book1.getName() + ", " +
book1.getAuthor() + ", " + book1.getPages() + " pages");
        System.out.println("Book 2: " + book2.getName() + ", " +
book2.getAuthor() + ", " + book2.getPages() + " pages");
    }
}
```

Spring Tools Suite- error

Tuesday, February 13, 2024 12:25 PM

Spring Tools Suite IDE-

Sprung initializr project window error-
SunCertPathBuilderException: unable to find valid certification path to requested target

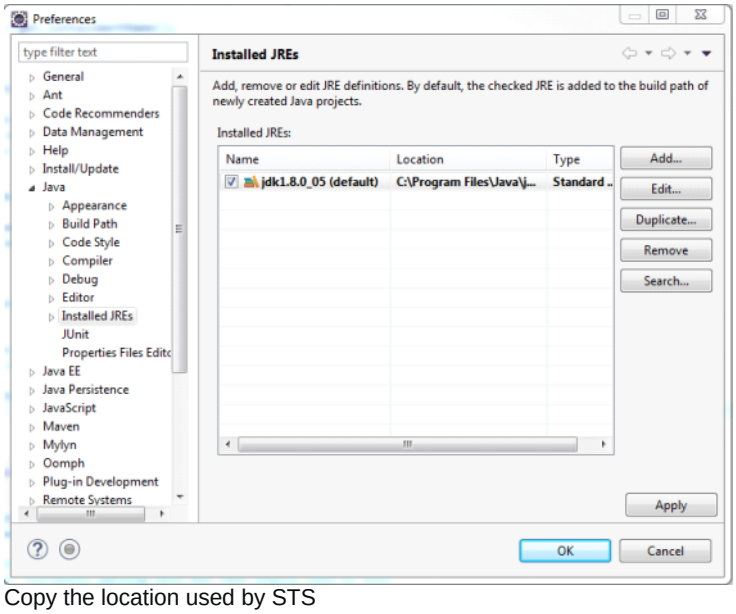
Solution:-

java - Spring tool suite- SunCertPathBuilderException: unable to find valid certification path to requested target - Stack Overflow

The following solution was given in the above article-

In Corporate Network, you may not have admin rights to import to cacerts file. The below steps helps to use STS in corporate network without admin privileges

1. Know your %userprofile% directory
   - Start -> Run
   - type %userprofile% and hit enter
   - This opens a folder for which you may have write access. Note the path
2. Export the corporate proxy certificate
   - On Chrome, go to https://start.spring.io
   - On the location bar click on the 'Lock' symbol next to https.
   - Select 'Certificate(Valid)' on the ensuing pop-up.
   - On the resulting dialog box, click on the 'Certificate Path' tab, from under certificate path tree select the root node, and then click on 'View Certificate'
   - On the resulting dialog box, click on the 'Details' tab and then click on 'Copy to File'
   - This brings up the 'Export wizard', click on 'Next'.
   - Leave the certificate format to default 'DER encoded..', click on 'Next'.
   - Provide CorpRootCA.cer as file name for the certificate and save in %userprofile%/cacerts folder. You may have to create the cacerts folder in your %userprofile%
   - Click Finish
3. Identify the JRE Path used by STS
   - Open STS
   - Click Window -> Preferences
   - Click Java -> Installed JRE



   - Copy the location used by STS
4. Copy cacerts file
   - Goto the jre location from Installed JRE in explorer
   - Goto lib/security and copy the cacerts file
   - Goto %userprofile%/cacerts and paste it inside the director
5. Adding corporate proxy certificate using keytool
   - Goto the jre location from Installed JRE in explorer
   - Type cmd in the url bar to open command prompt at the location

- ./bin/keytool -import -alias CorpProxy -keystore %userprofile%/cacerts/cacerts -file %userprofile%/cacerts/CorpRootCA.cer -storepass changeit
  The 'storepass' should be 'changeit' which is the default password. If changed use the correct storepass

6. Making STS point to new certstore
   - Open the folder which contains the SpringToolSuite executable
   - There will be a configuration file with same file as executable. In my case it was SpringToolSuite4.ini. Open the file
   - Append the below lines after changing the %userprofileto match the user's directory. Update trustStorePassword as required. changeit is the default value

   -Djavax.net.ssl.trustStore=%userprofile%\cacerts\cacerts
   -Djava.net.ssl.trustStorePassword=changeit
7. Restart STS and it should no longer throw
   SunCertPathBuilderException: unable to find valid certification path to requested target

---

Part 1->Learning Java IDEs- IntelliJ IDEA

Tuesday, July 4, 2023 11:28 AM

IntelliJ IDEA-

jetbrains.com
jetbrains develops and manages IntelliJ IDEA

Community edition- Open source, licensed under Apache 2.0, JVM and android development

Ultimate edition- additional tools and features for web and enterprise development, distributed with 30-day trial period.

Create new project- Select the type of project (Java, Groovy, Kotlin, Maven, Gradle, Java FX) -> Select JDK version -> Next -> Project name-> Project location can be changed here

Change font size with Ctrl +mouse wheel -
File-> Settings-> Editor-> General-> Change font size with mouse wheel

New package-
src-> right click-> New -> Package

Right click in the code-> Generate-> Constructors, Getter setters

File-> Settings -> Keymap

- Keymap has keyboard shortcuts. We can select the IDEs there. If we select Eclipse, eclipse's keyboard shortcuts could be used in IntelliJ

Plugin-
File-> Settings-> Plugin

He installed a plugin called Presentation Assistant which pops up with a keyboard shortcut on screen once we select any option from IntelliJ or click on something, so by seeing that keyboard shortcut in pop-up we can remember the keyboard shortcut.

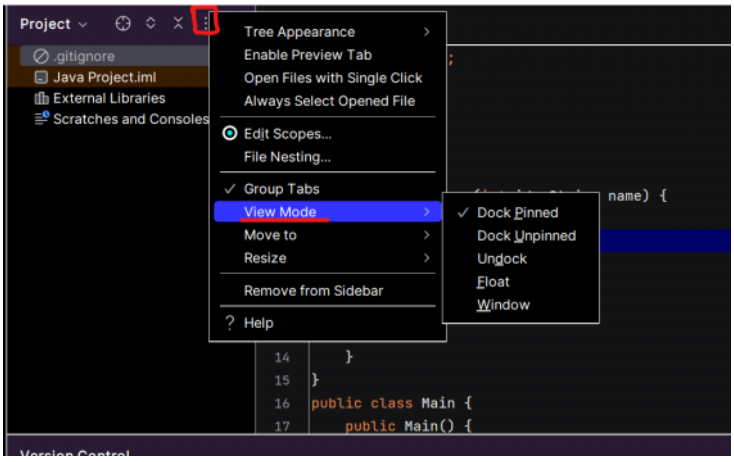
Theme changing - Light, Dark, High Contrast
File-> Settings-> Appearance and Behavior-> Appearance-> Theme

Theme can be installed from plugins.jetbrains.com-> search for a theme-> Install to IDE

For annotations to work in IntelliJ-
File-> Settings-> Build, Execution, Deployment-> Compiler-> Annotation Processors-> Enable annotation processing

Close project in IntelliJ-
File-> Close project

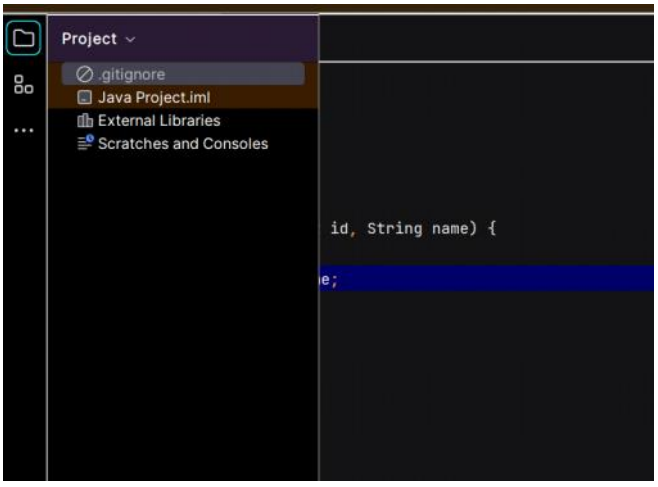
View mode-



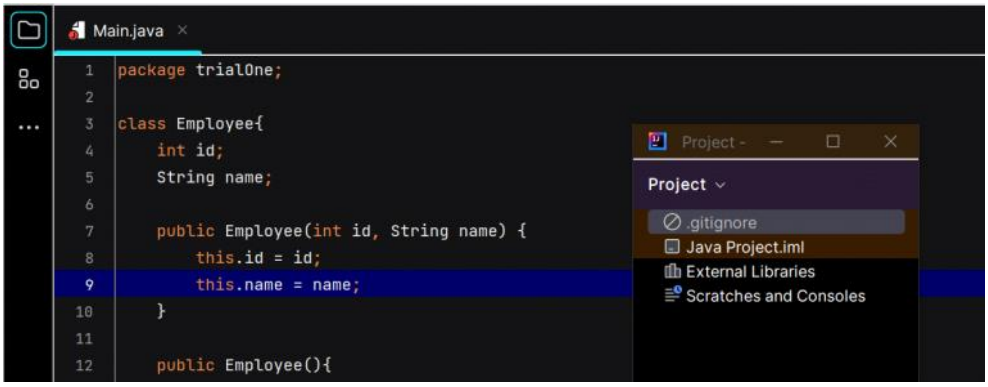
Go to project's window.

View mode- Dock pinned (window will be always visible)

- Dock unpinned(window will be visible only when you're there. If you switch the window, it will be gone)
- Undock (project's window will open over editor's window)
- Float (the window can be moved)
- Window (the window can be moved on other screen also)



This is undock



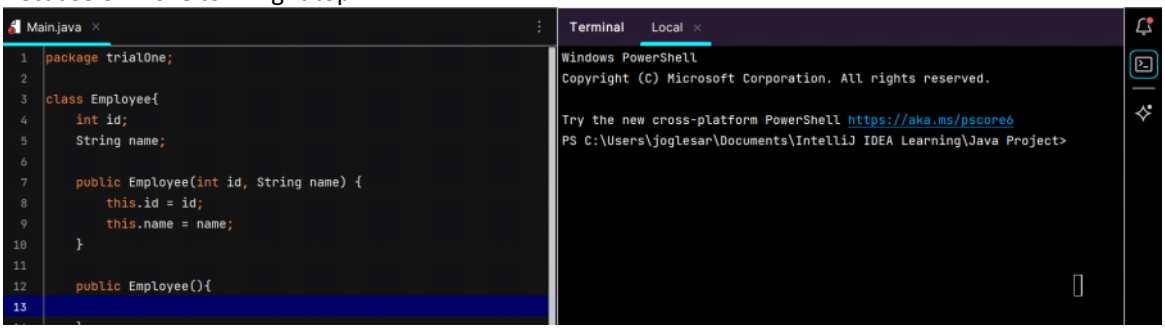
This is window.

Window-> Layouts-> Save changes in current layout

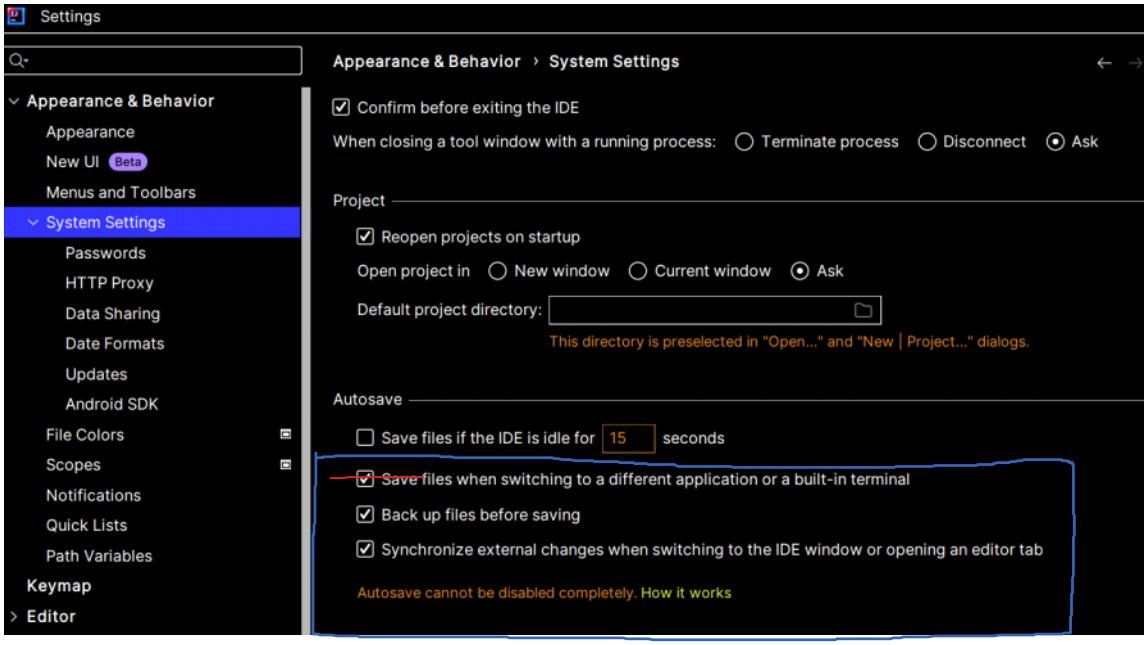
Move to

Click on the icon-> 3 dots-> Move to->

Because of Move to-> Right top



Autosave
File-> Appearance and behavior-> System settings



Move line, statement-

Code-> Move line down, Move line up (put the cursor on the line to move and press the shortcut) (Alt+Shift+up/down arrow)

Code-> Move statement down, Move statement up

---

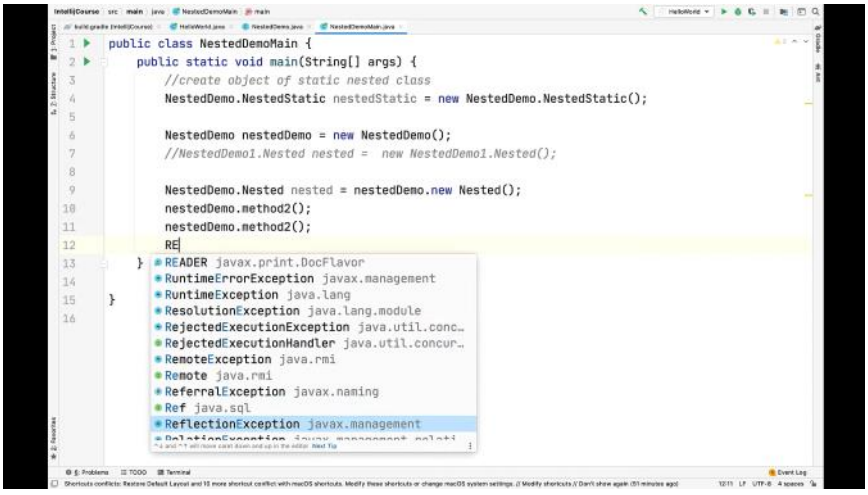
for loop generation

In java class, inside a method- initialize a variable like- int i=0;

Now in the next line if you type- i. you'll get some options, one of them will be for loop

---

Camel Hump Technique



In order to use any class in code-

Type the capital letters from that class

For class **R**untime**E**xception.java -> Type RE in the code (capital letters of class)

Changing method signature

If I've written a method-

public void print(){ }

Now if I want to change this as public void print(int a){ }

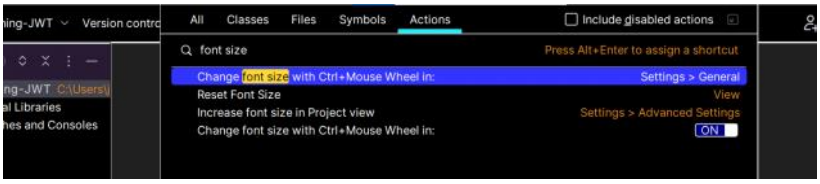
Then at all the method calls, the signature has to be changed.

Put the cursor on the method.

Refactor menu-> Change signature. There we can put the default value also.

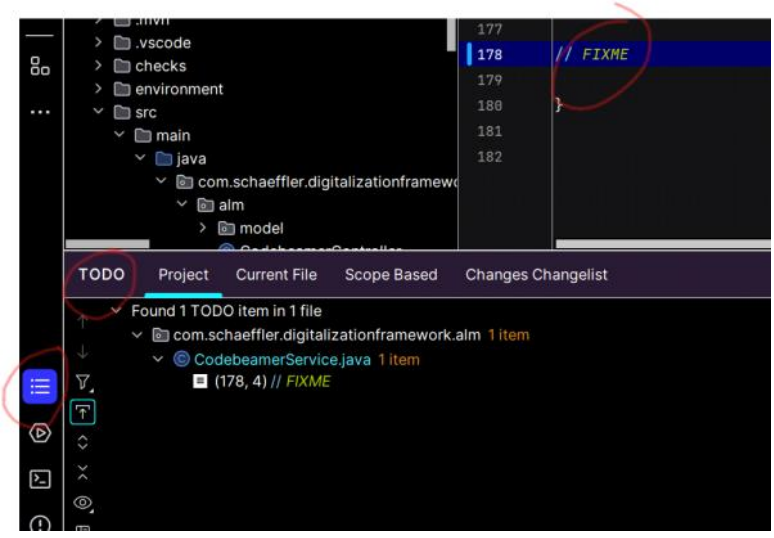
Double shift- Shift+Shift- search everywhere within project

Find action- Ctrl+ Shift+ A



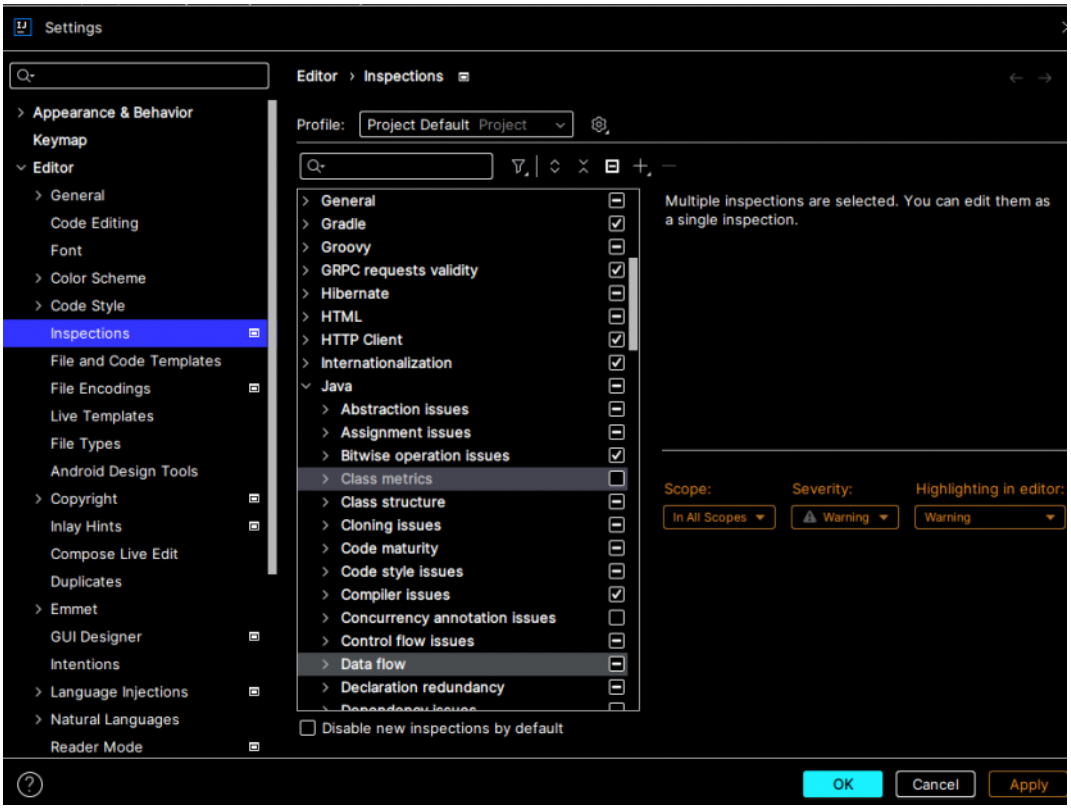
Navigate menu-> Class

To create a to-do item in IntelliJ IDEA :-
//TODO
//FIXME
are in IntelliJ and TODO item is created

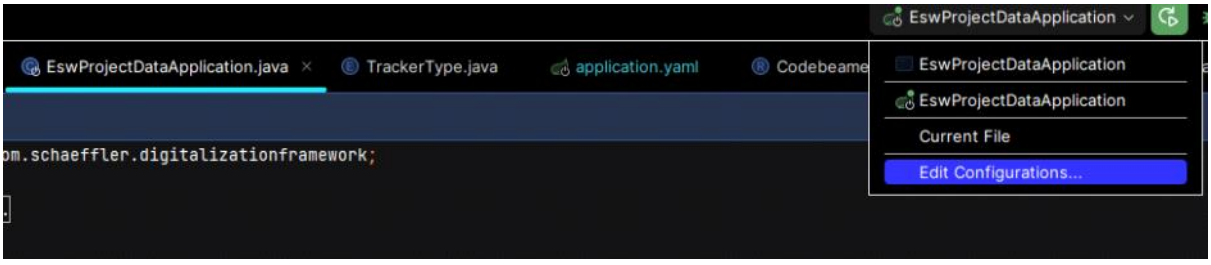


To provide indentation before the lines of code-> **Select the lines in the code-> press Tab**
To move the lines of code to left or to remove indentation- **Select the lines in the code-> press Shift+Tab**

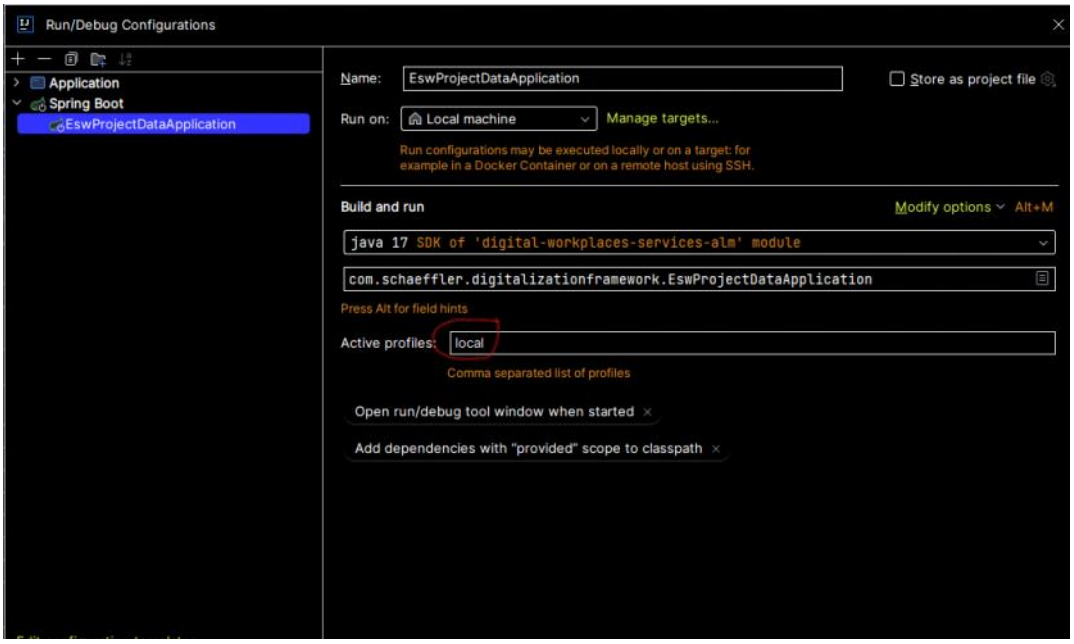
For checking Java coding guidelines in IntelliJ IDEA->
File-> Settings-> Editor-> Inspections



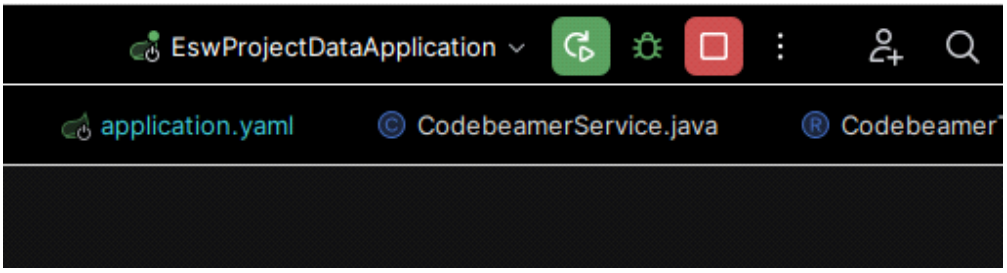
This setting helped Git to work in my IntelliJ IDE->
File -> Settings-> Search proxy - Enable- Autodetect proxy settings
File-> Settings-> Search Certificates- Add certificate (security certificate)

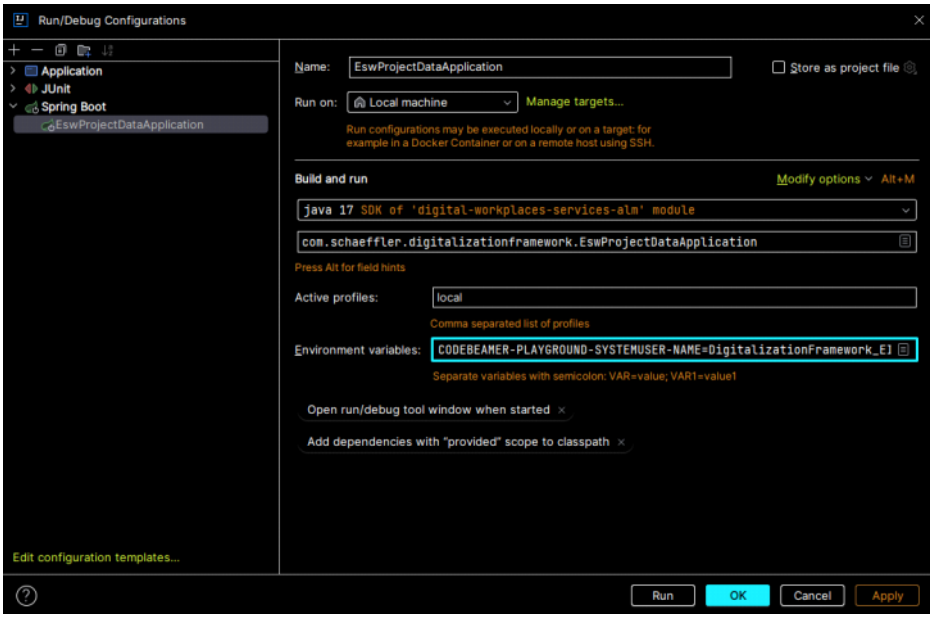


Edit configurations-> Active profile- local

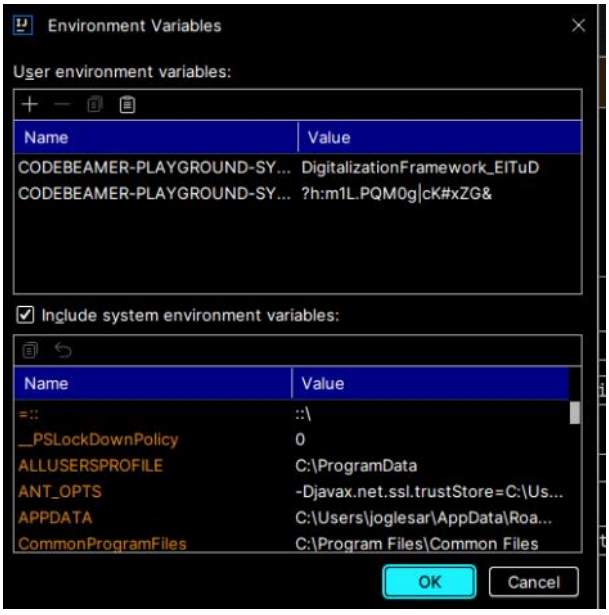


To rerun the code, in the top right corner, comes an option.

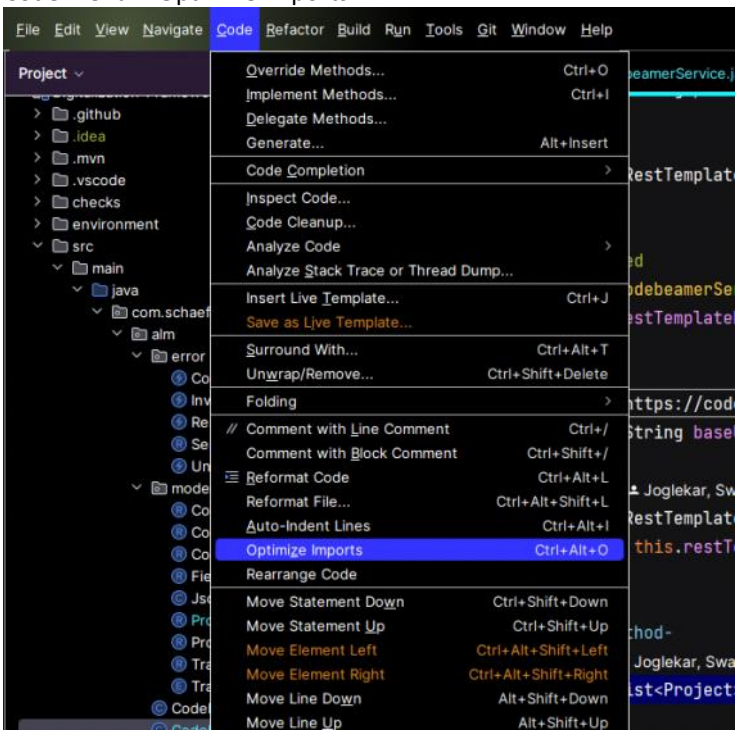




In environment variables- added username and password; separated by semicolon(;)



Optimize Imports-
Code menu-> Optimize imports

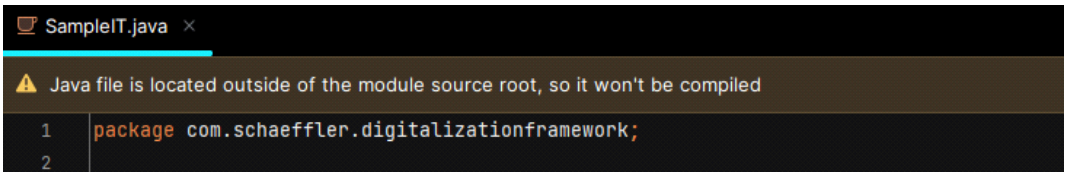


The dependency that I had added in pom.xml file in Springboot project was not working and was red in color.
Right click on the file-> Reload

Powershell setup to make it visible in IntelliJ idea-

1. Set the path in system variable C:\Windows\SysWOW64\WindowsPowerShell\v1.0 (I set this one)
2. Restart the computer

'Java file is located outside of the module source root, so it won't be compiled'



In most Java projects using an IDE such as IntelliJ or Eclipse, the source files (such as .java files) are placed in a specific directory structure, typically under a "src" directory. This directory is typically marked as a source root in the project's build configuration, so that the compiler knows where to find the source files.

If you create a new Java file in a directory that is outside of the source root directory, the IDE may not recognize it as part of the project's source code and will give you the "Java file is located out of module source root" error message.

Find and replace in intellij-
Select text and use Ctrl+R.
Replace and Replace All options.
Write what you need to find. Then, write what you need to replace that with. Then click on replace or replace all.

What to learn in IntelliJ?

1. Creating Java project-- Done
2. Increasing font size of code by mouse scroll-- Done
3. Creating java packages-- Done
4. **Git with intellij**
5. **Debugging with intellij**
6. Installing plugins--Done
7. Importing a pre-created Maven project
8. Searching a keyword in all files and changing that keyword for all files
9. Generate getter setters, constructors-- Done
10. How to create springboot project and add dependencies in intellij

Part 2->Learning Java IDEs- IntelliJ IDEA

Thursday, April 18, 2024 3:03 PM

1) **A project created in eclipse- .zip file-> to be imported in intelliJ-> Process:**

File->New-> Project from existing sources-> Import project from external model-> eclipse-> few times click next-> select jdk-> Ok

Marking the test folder- test sources root:
src-> test-> right click-> mark directory as-> test sources root

By marking the directory as a test sources root, IntelliJ can then appropriately configure the build process to compile and run the tests separately from the application code.

Similarly as above, mark the 'resources' folder as 'test resources root'.

---

2) **How to create new test in IntelliJ?**
select class name inside the file-> Ctrl+Shift+T-> Create new test-> Testing library- JUnit5 -> Ok
If you get an error, Add JUnit5 to classpath

---

3) How I solved the error- Source release 17 requires target release 17 ?

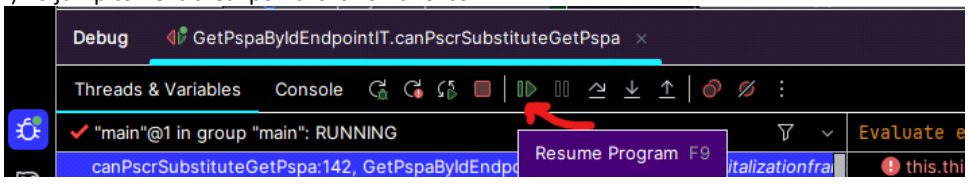
- It happens when we're using older version of compiler to compile newer version of source code.

File-> project structure-> Project sdk 17,
Modules-> Sources-> Lang. level 17
Modules-> Dependencies-> Module sdk-> project sdk 17-> Check the box Junit 5.8

---

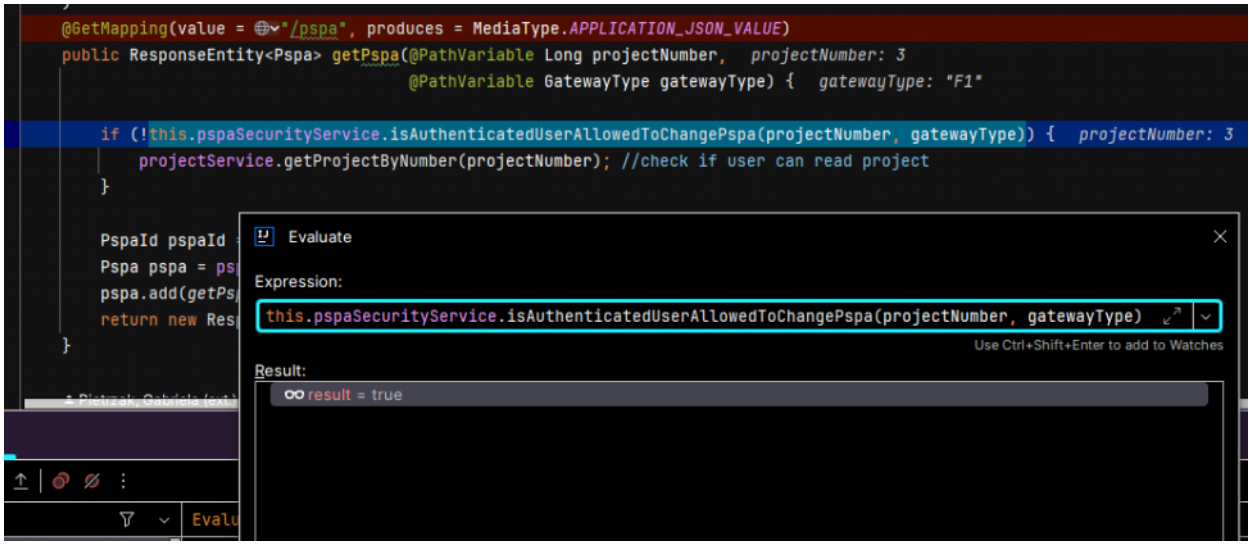
4) **Debugging in IntelliJ**

i) To jump to next breakpoint- click on this icon



---

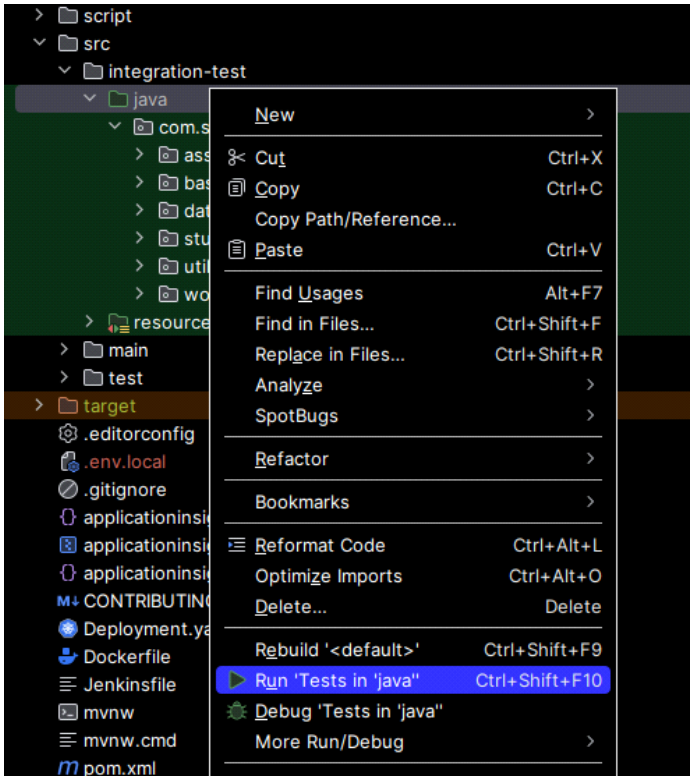
ii) to evaluate an expression while debugging- for e.g. to know the value of expression inside if brackets:
Select the expression and right click-> Evaluate expression-> Click evaluate



---

5) How to run all tests inside a folder?

java folder-> right click-> Run tests in java

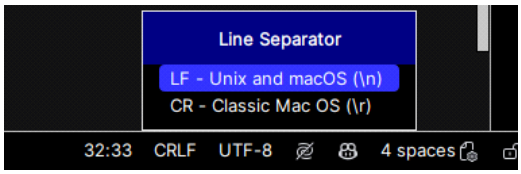


CRLF, LF settings in Intellij

CRLF- Carriage Return Line Feed

In general, Unix/Linux systems use LF (\n) as the line ending character in text files.
On the other hand, Windows systems use CRLF (\r\n) as the line ending character in text files.
It is important to use consistent line endings throughout each file, as inconsistent use of line endings may cause unexpected behavior in the execution of the code.

Bottom right corner of IDE. Has to be done for every file


Line Separator
LF - Unix and macOS (\n)
CR - Classic Mac OS (\r)
32:33
CRLF
UTF-8
4 spaces

Part 3-> IntelliJ Idea learnings

Wednesday, August 28, 2024 3:35 PM

Shortcuts:

1) In IntelliJ
Ctrl+ Z

Ctrl+Shift+F -> Find in files

Ctrl+Shift+R -> Replace

Ctrl+ Shift+N-

Ctrl + G - go to specific line number
If we write 42:1 > then its 1st character on line 42

select name and use Ctrl+ Click - goes to wherever that name is used/called in the code

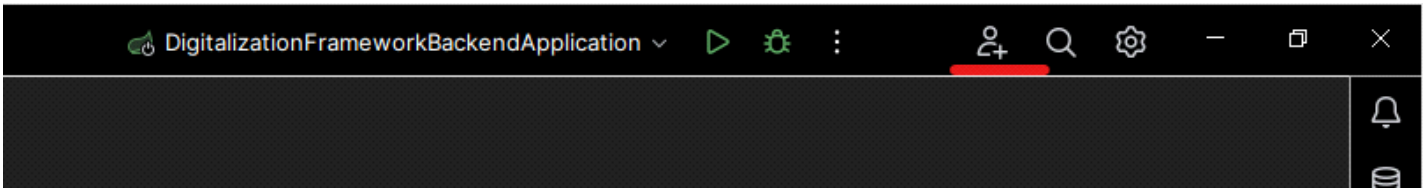
While debugging the application- to debug next line- Fn(Function) + F8

IntelliJ Idea features

Wednesday, January 10, 2024 2:17 PM

IntelliJ Idea Ultimate-

1. "Code With Me" feature:



Start session
or if other person has already sent the link of session, click on it and IntelliJ will start downloading a client.
If the session is on, you will be able to see other person as well. There is also an option to talk to other person.

According to my team member- Piotr, Code with Me feature should not be used because data passes via Jetbrains server.
_____

Explained by Oliver

Friday, September 1, 2023 6:31 PM

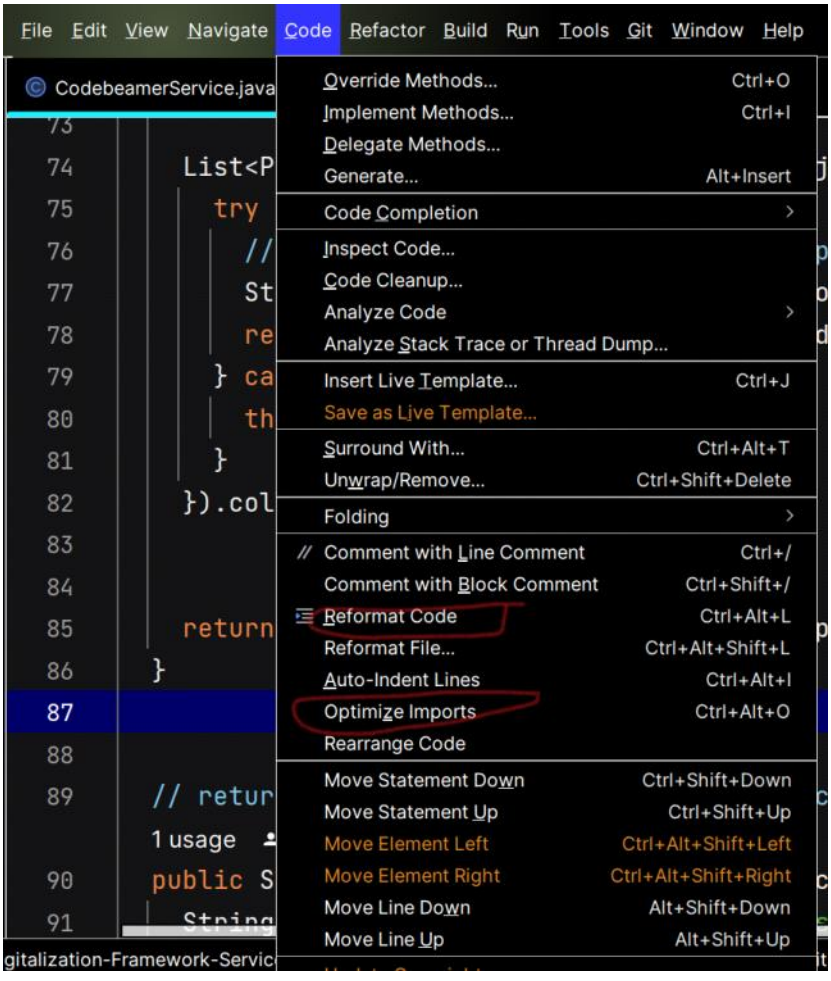
1) File-> Settings-> Keymap-> search- Format code

2) **Macro recording**

Edit-> Macros-> Start Macro recording

Code-> Reformat code, Optimize imports

Edit-> Macros-> Stop Macro recording



If you've saved the macro, you can find it in File-> Settings-> Keymap-> Macro

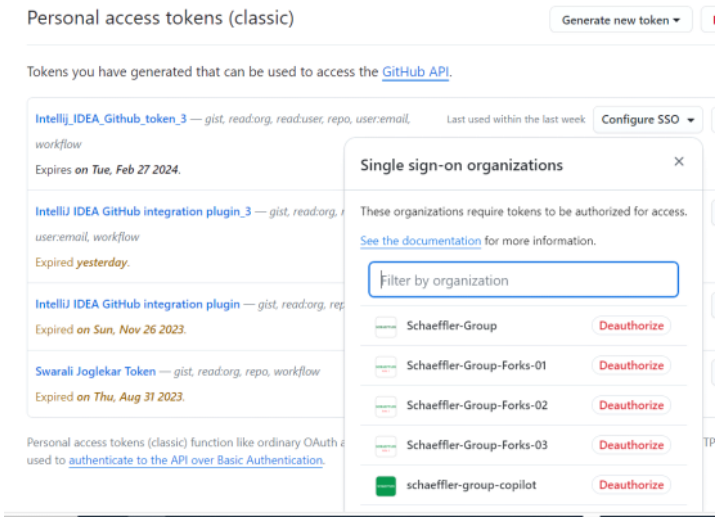
Git with IntelliJ IDEA

Friday, July 21, 2023 2:41 PM

Git with IntelliJ IDEA

Connecting IntelliJ IDEA with GitHub

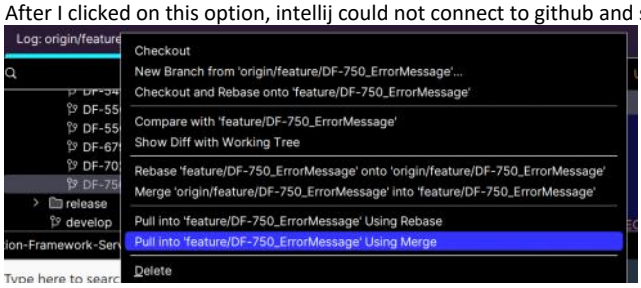
1. Either intellij idea will ask you to generate the token or go to github developer settings and generate the token by selecting required checks. Select the time period for token to be valid.
2. Authorize Schaeffler Group for required token
3. Use the token in intelliJ to connect to github.



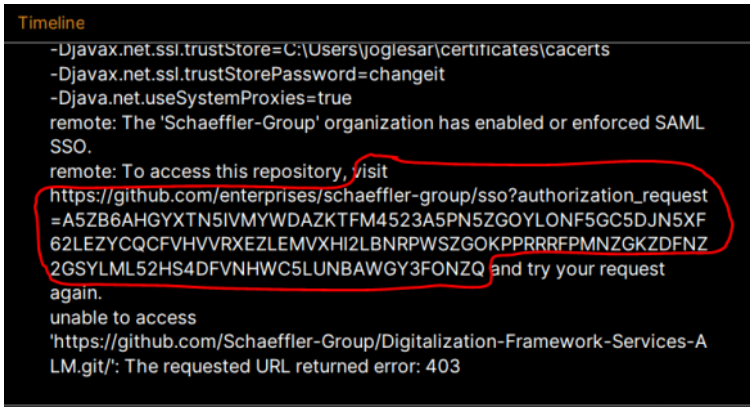
4. Make sure that following settings are done.

File -> Settings-> Search "proxy" - Enable- Autodetect proxy settings
File-> Settings-> Search "Certificates"- Add certificate (security certificate)

I still had problem to connect to github.
After I clicked on this option, intellij could not connect to github and sent me a notification.

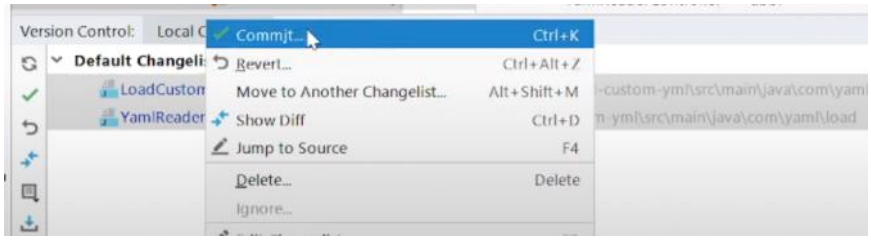


In IntelliJ I had got following notification for update failed. When I authorized Schaeffler Group using this url, I got connected to GitHub.



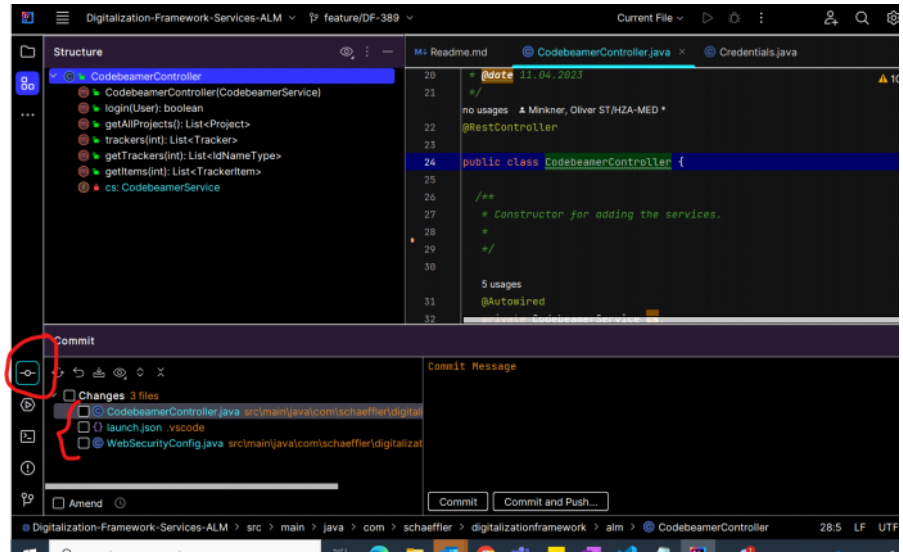
Using Git in IntelliJ :-

1) File-> New-> Project from version control-> Put the URL-> Clone
2) Open Version Control in IntelliJ. Either using search in top right corner or directly using version control in the bottom of IntelliJ
3) Select the files-> Right click-> Show diff(difference in previous changes), Commit options can be used.

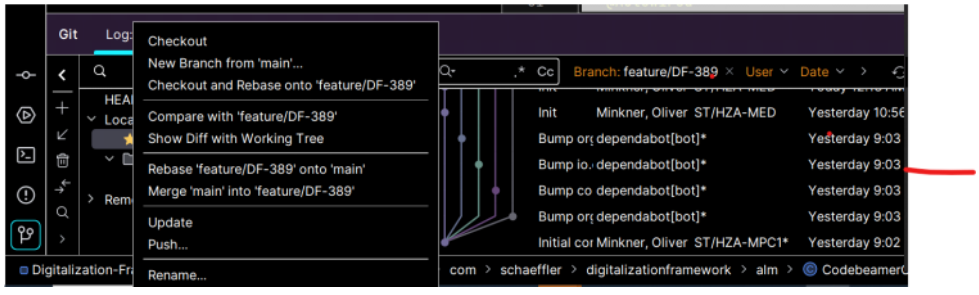


4) VCS menu-> Git-> Push

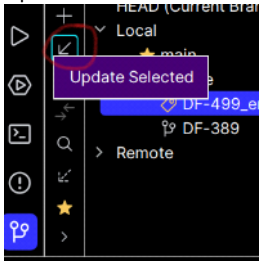
In my IntelliJ-



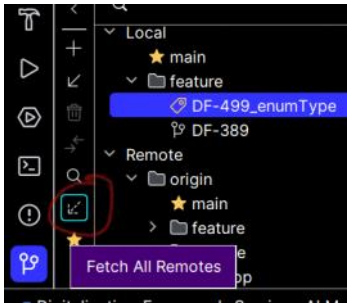
Right click on branch name-



Update selected-

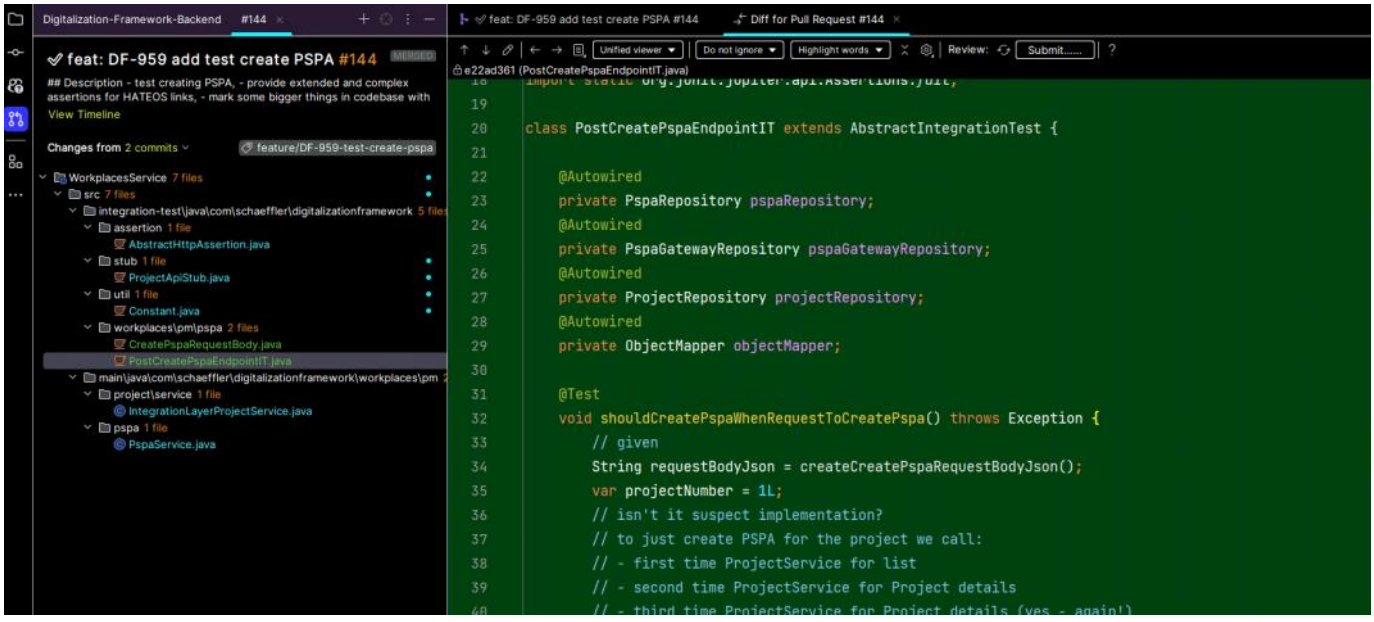


Fetch all remotes-



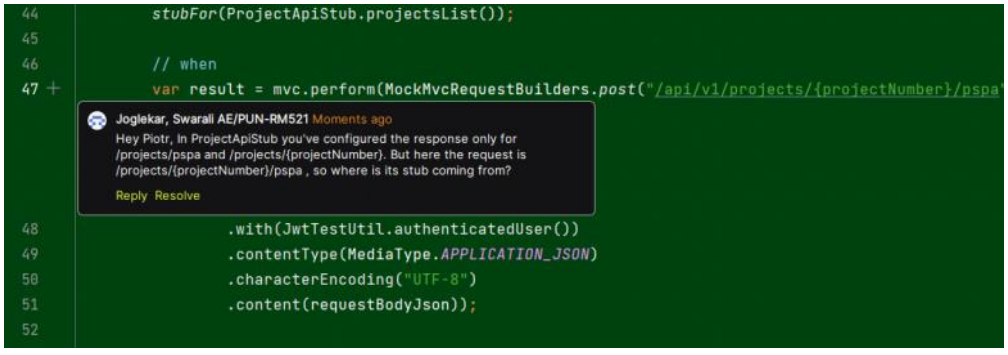
Once you checkout to feature branch, if you right click on develop, you get the option to merge that feature branch to develop branch.

Viewing Pull request from IntelliJ idea:



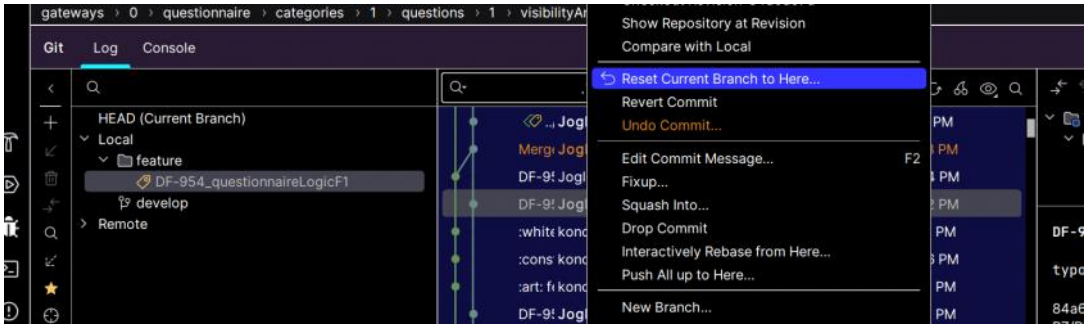
There's a filter to choose open and closed pull request and the author.

Adding comments in changes in pull request:



+ symbol appears on hovering over the line.

Undo commit in IntelliJ Idea:-



Open the Git tool window. Click on the "Log" tab to see the commit history.
Right-click on the commit you want to undo and select "Reset Current Branch to Here".
In the popup dialog, select the "Soft" option to keep your changes in the working directory.
Click "Reset" to undo the commit.

This only has effect in local repository.

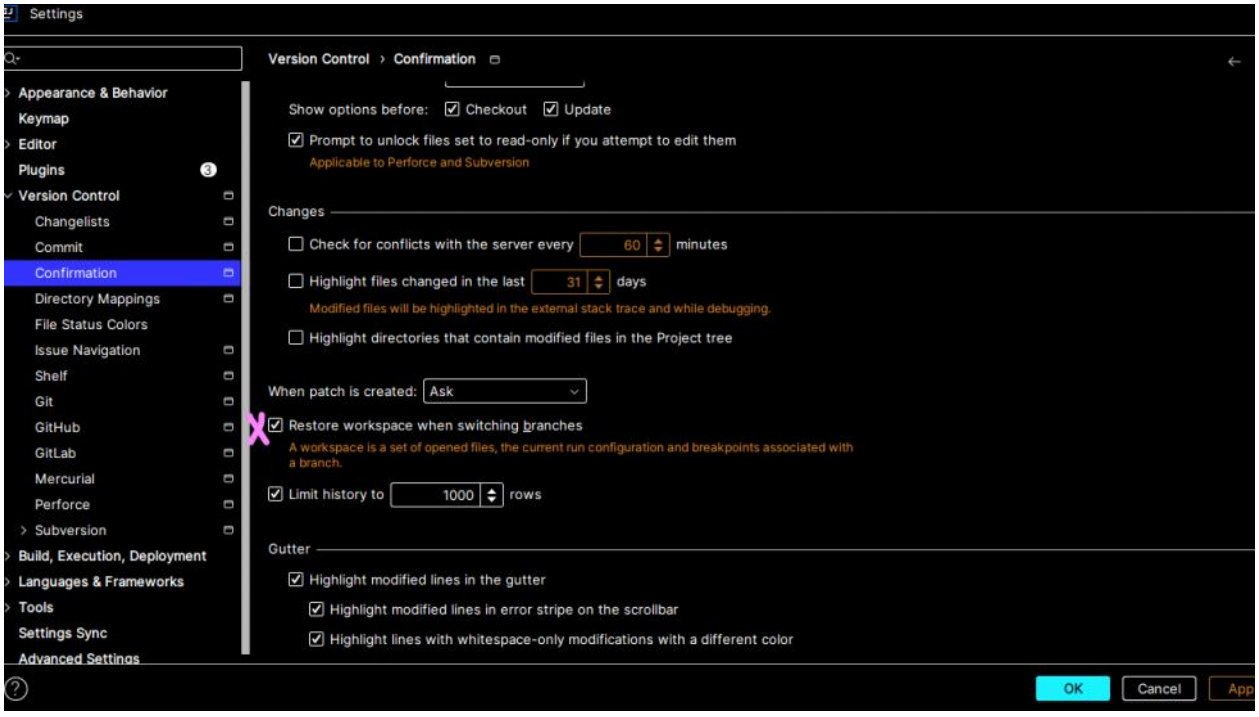
Difference between- 'Reset current branch to here' and 'Revert commit'

When you use "**reset current branch to here**", you're basically deleting all commits after the selected commit and starting fresh from there, which means any changes made in the deleted commits will be lost forever.
When you use "**revert commit**", you're keeping all the commits in the branch history, but undoing the changes made in the selected commit by creating a new commit to undo those changes. This means the changes made in the reverted commit are still kept in the branch history, and you can continue making new commits from that point onwards.

The way to revert pushed change->
After using revert commit, we can push again.

I had done some changes in Edit configurations in development branch. But when I switched to another branch, they were gone. I got a notification for branch workspace settings.

Branch workspace settings-> Remove the checkmark of restoration- So that the workspace will remain the same for all branches

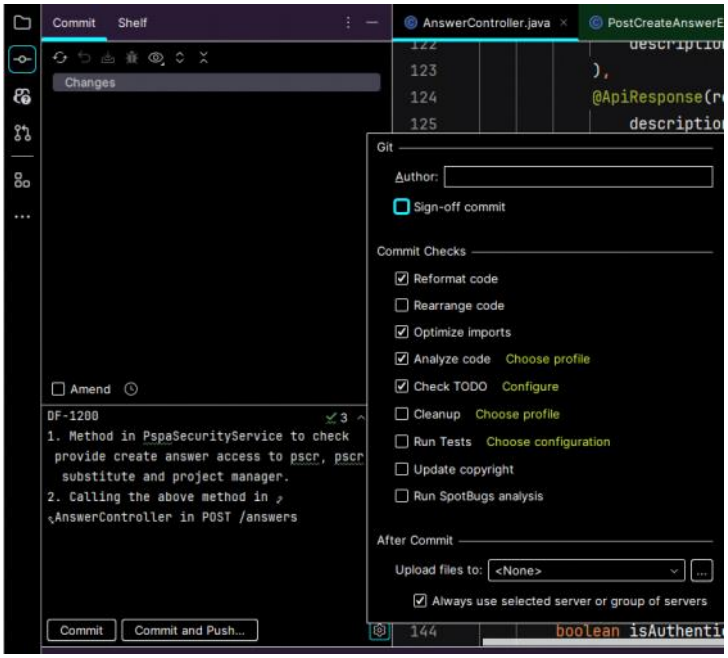


How to merge develop branch in feature branch?

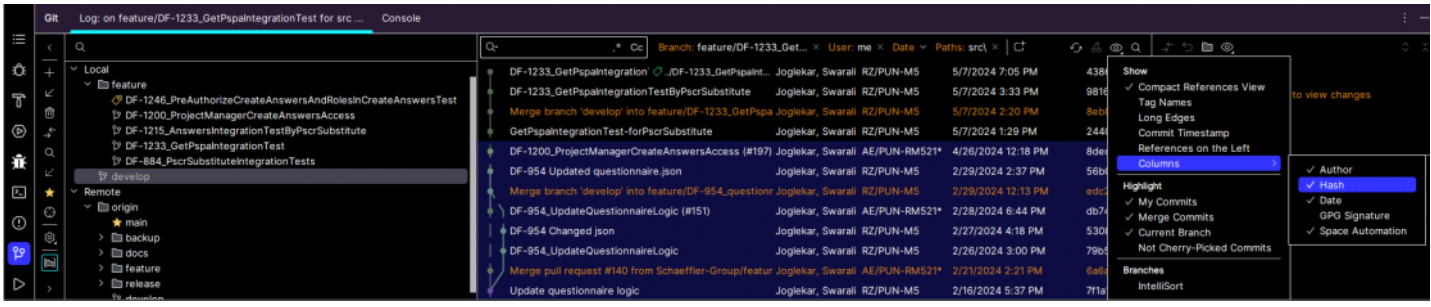
- Pull the changes in develop branch.
- Check-out to feature branch.
- In IDE like IntelliJ, if you right click on develop, you'll get the option to merge develop branch in feature branch to which you've checked out.

Settings before commit in IntelliJ IDEA

Click on settings icon (in the image- next to commit and push option)
While making the commit in IntelliJ idea, the following settings will be checked. e.g. checkbox of reformat code

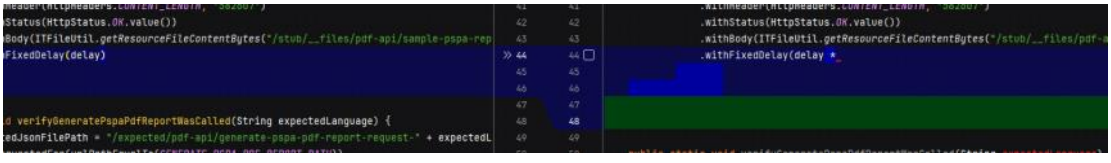


How to view Hash of commit in IntelliJ Idea?



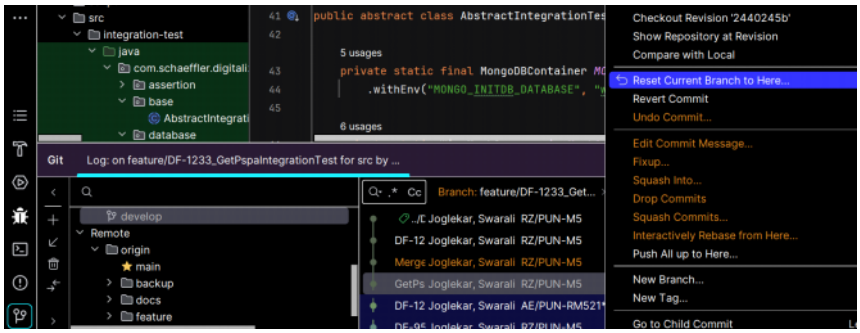
eye symbol in the middle window-> Columns-> Check Hash

Blue, Green and Red areas in IntelliJ IDEA- during comparison of files in Git
Blue-> modified lines in file
Green-> new lines in file
Red-> removed lines in file



Locally- **hard reset** in IntelliJ idea-

Right Click on the previous commit which you want to set as the recent commit
-> Reset Current branch to here-> Hard reset



---

right click-> Git -> **Show history**

Force checkout- checkout and overwrite local changes
Smart checkout- Shelve local changes, Checkout, Unshelve

You have to resolve Patch conflict after smart checkout.

Need of **stash**-

- When you need to switch to another branch to work on a different task but haven't finished your current work.
- Stashing your changes allows you to switch branches without committing or discarding your work.

Stash changes in Branch 1- Saves your uncommitted changes in Branch 1 to a temporary stash.
You can now safely switch to Branch 2 without losing your work.
Make your desired changes on Branch 2.
Switch back to Branch 1.
Unstash changes in Branch 1- This applies the stashed changes back to Branch 1.

Git menu in IntelliJ-> Uncommitted changes- Stash changes

You can't stash specific file. Stash will applied to your whole working directory.

Patch- local development, collaboration, and applying changes from external sources.

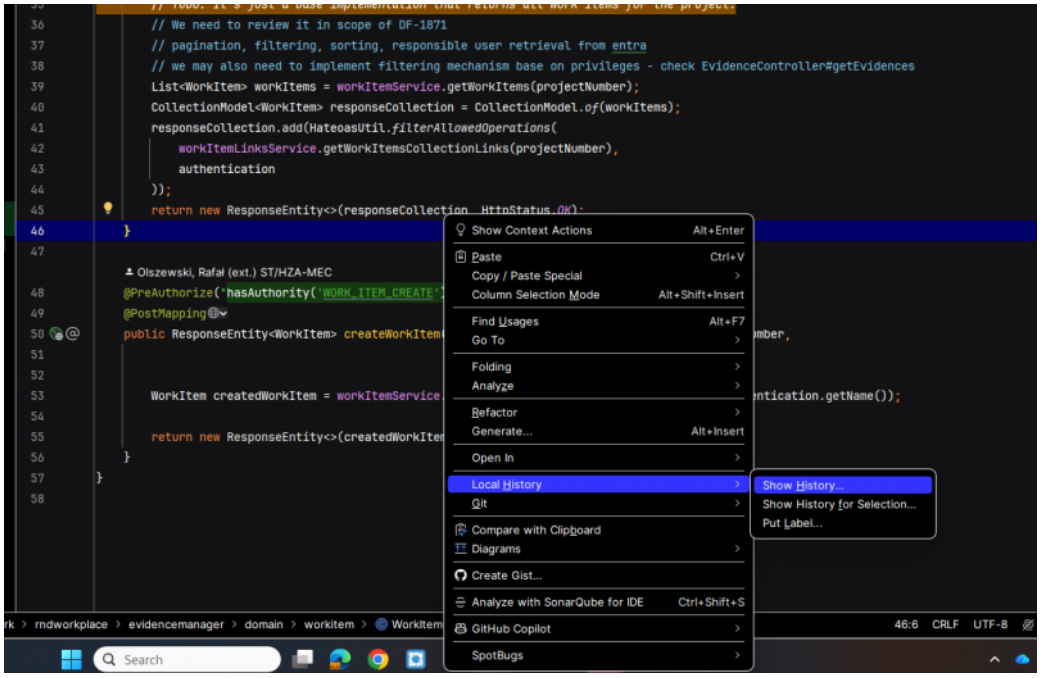
Create patch- specify patch file name and location.
Git menu-> Patch-> Create patch from local changes
You can select files there. You have to write a message.

Apply patch- select the patch file you want to apply.
Git menu-> Patch-> Apply patch

We can see all the changes we did in this file using Git operations. Even if we locally rollback all the changes, we can still see them here

In IntelliJ-

Local history-> Show history

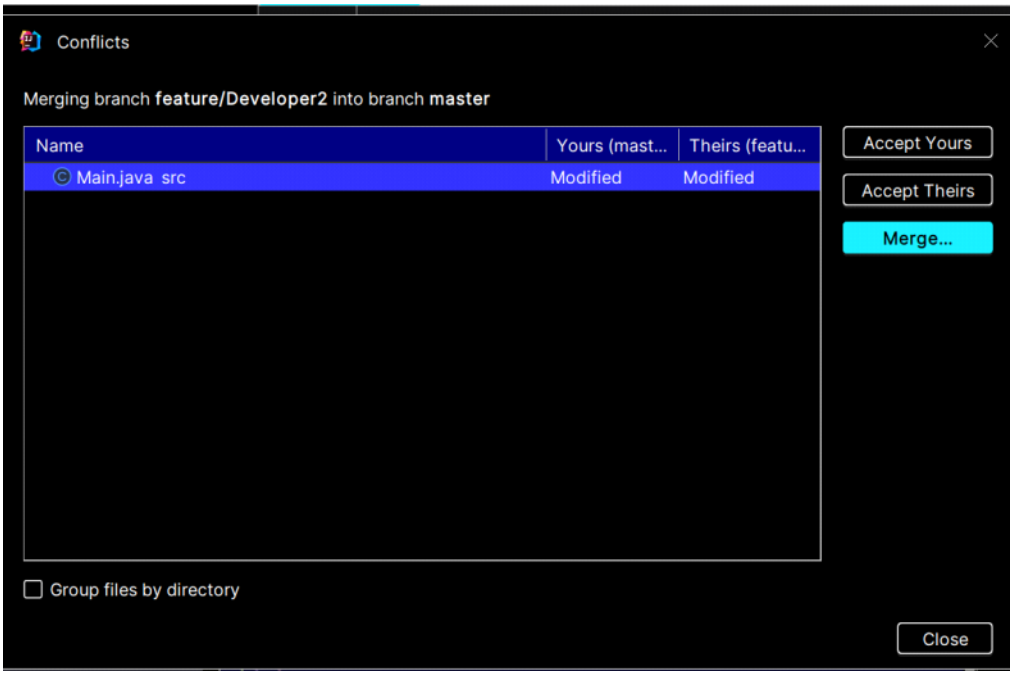

// We need to review it in scope of DF-1871
// pagination, filtering, sorting, responsible user retrieval from entra
// we may also need to implement filtering mechanism base on privileges - check EvidenceController#getEvidences
List<WorkItem> workItems = workItemService.getWorkItems(projectNumber);
CollectionModel<WorkItem> responseCollection = CollectionModel.of(workItems);
responseCollection.add(HateoasUtil.filterAllowedOperations(
workItemLinksService.getWorkItemsCollectionLinks(projectNumber),
authentication
));
return new ResponseEntity<>(responseCollection, HttpStatus.OK);
}
Olszewski, Rafał (ext.) ST/HZA-MEC
@PreAuthorize("hasAuthority('WORK_ITEM_CREATE')")
@PostMapping
public ResponseEntity<WorkItem> createWorkItem
WorkItem createdWorkItem = workItemService.
return new ResponseEntity<>(createdWorkIte
}
}
Show Context Actions Alt+Enter
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Go To
Folding
Analyze
Refactor
Generate... Alt+Insert
Open In
Local History
Git
Compare with Clipboard
Diagrams
Create Gist...
Analyze with SonarQube for IDE Ctrl+Shift+S
GitHub Copilot
SpotBugs
Show History...
Show History for Selection...
Put Label...
rndworkplace
evidencemanager
domain
workitem
WorkItem
46:6
CRLF
UTF-8
Search

Solving Merge conflict in IntelliJ IDEA

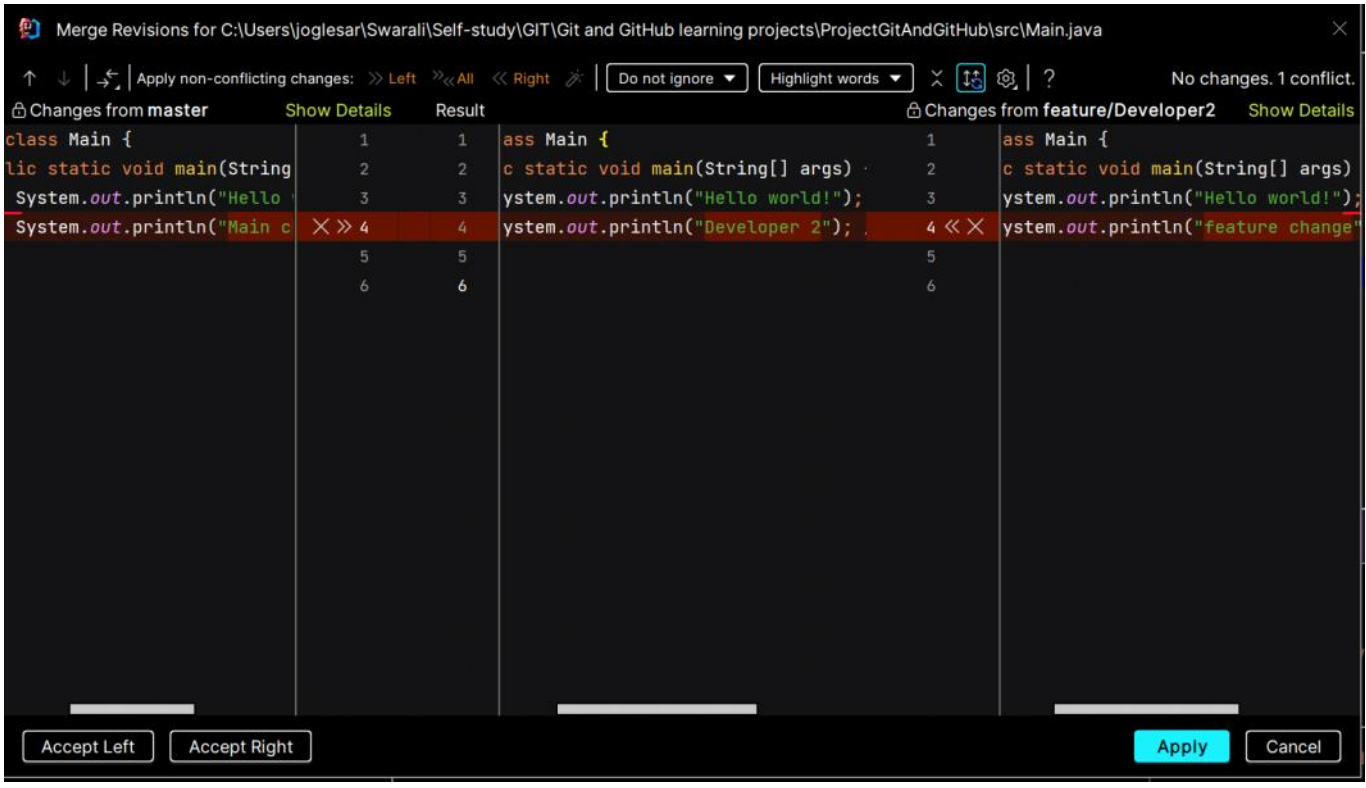
Sunday, February 4, 2024 10:44 PM

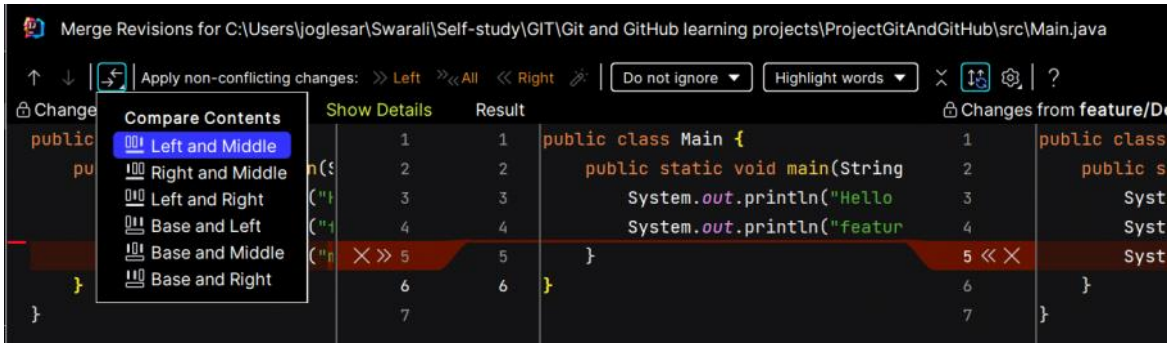
1)

IntelliJ IDEA IDE- During merge conflict:-
It initially showed me file name and options as Accept yours, Accept theirs, Merge.
I double clicked on file that had conflict.



2) Observe 'Show Details' options on both sides. It shows us the branch and its previous commits.
Remember >> and << symbols on both sides. Remember 'Accept left' and 'Accept right'.

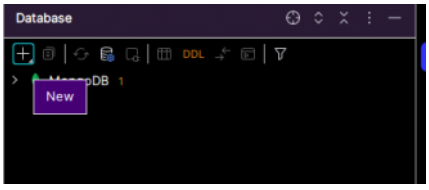

Merge Revisions for C:\Users\joglesar\Swarali\Self-study\GIT\Git and GitHub learning projects\ProjectGitAndGitHub\src\Main.java
Apply non-conflicting changes: Left All Right
Do not ignore
Highlight words
No changes. 1 conflict.
Changes from master
Show Details
Result
Changes from feature/Developer2
Show Details
Accept Left
Accept Right
Apply
Cancel


Merge Revisions for C:\Users\joglesar\Swarali\Self-study\GIT\Git and GitHub learning projects\ProjectGitAndGitHub\src\Main.java
Apply non-conflicting changes: Left All Right
Do not ignore
Highlight words
Compare Contents
Left and Middle
Right and Middle
Left and Right
Base and Left
Base and Middle
Base and Right
Show Details
Result

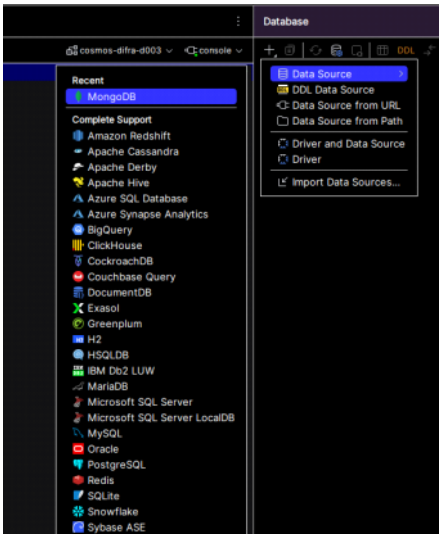
IntelliJ and MongoDB setup for project

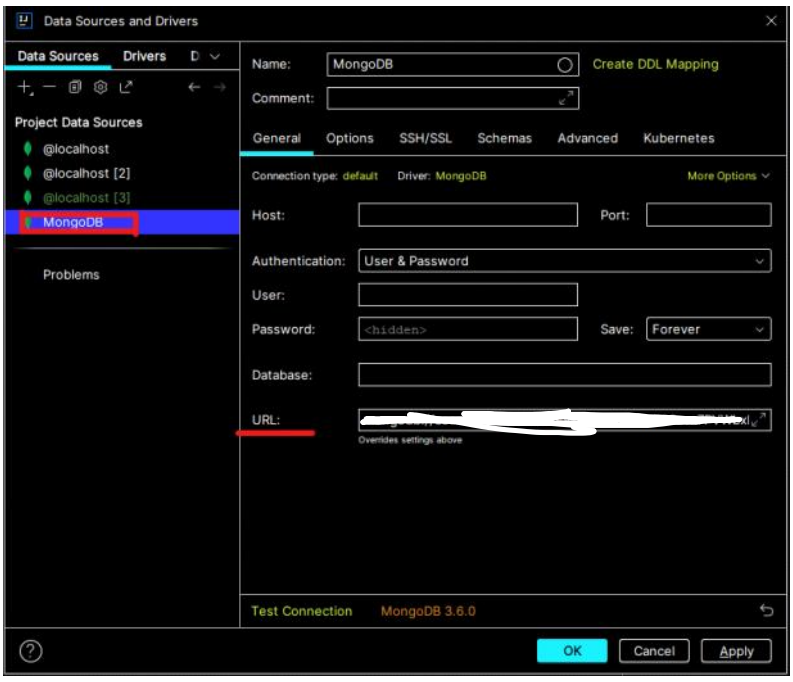
Monday, January 22, 2024 5:12 PM

Click on database symbol in IntelliJ.



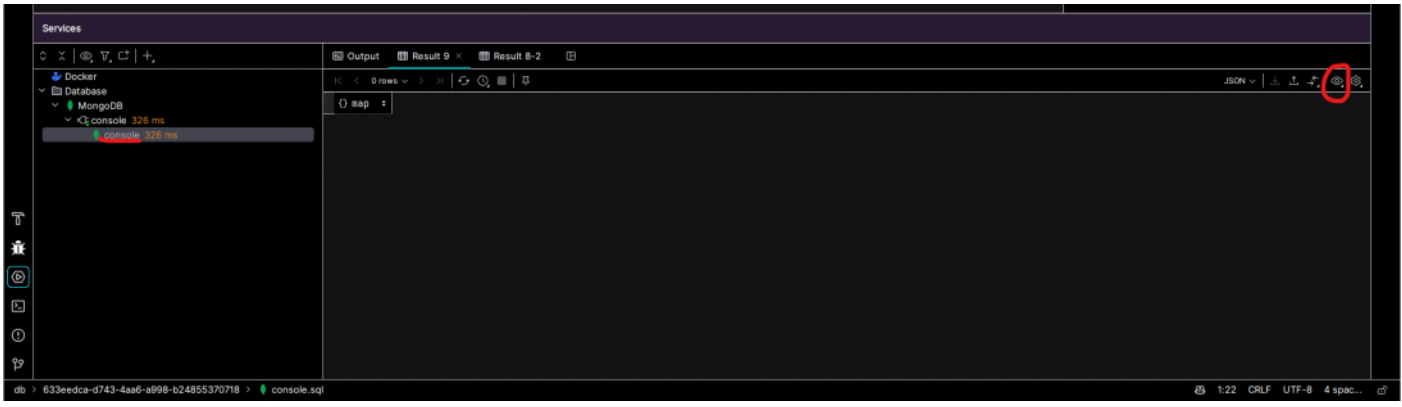
Click on + -> Data source -> MongoDB





To connect to remote database mongodb, put the url in mongodb section.
If you want local mongodb to work, you need docker setup.

For PSPA project-> The URL is->



Database queries->
db.userproject.find()
db.project.find()

Following commands to delete PSPA

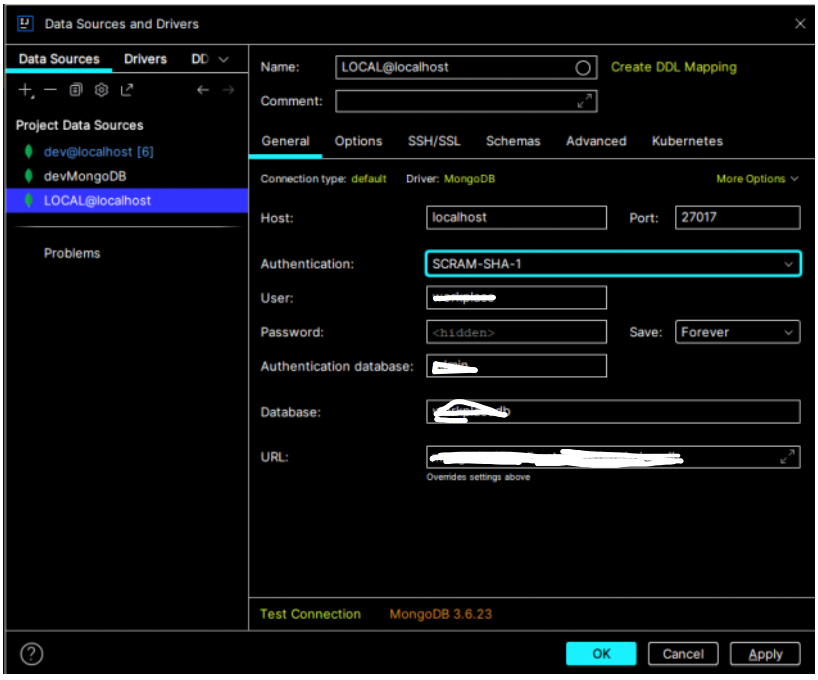
db.project.drop()
db.userproject.drop()
db.pspa.drop()
db.pspagateway.drop()
db.answer.drop()

To execute all the commands together, select all the lines and click on execute.

db.getCollectionNames() - to get the name of collections

Local mongodb setup:-

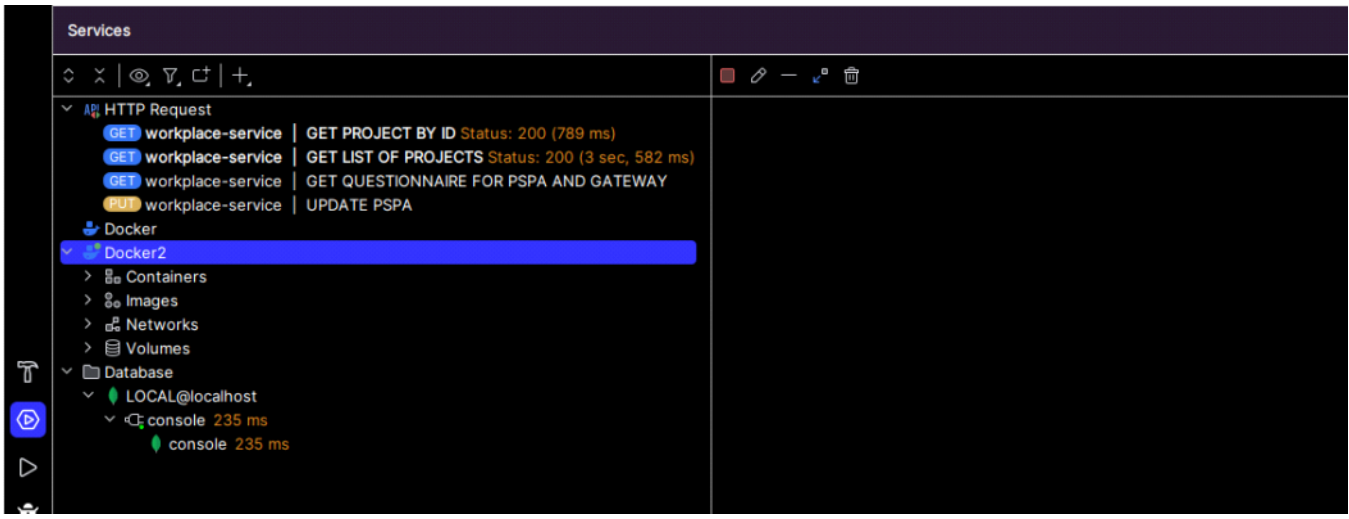
Authentication- SCRAM-SHA-1
put user and password.
password is hidden.



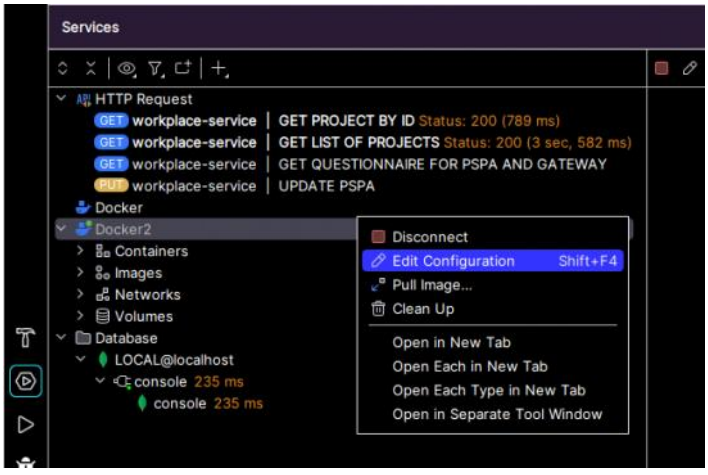
IntelliJ and docker

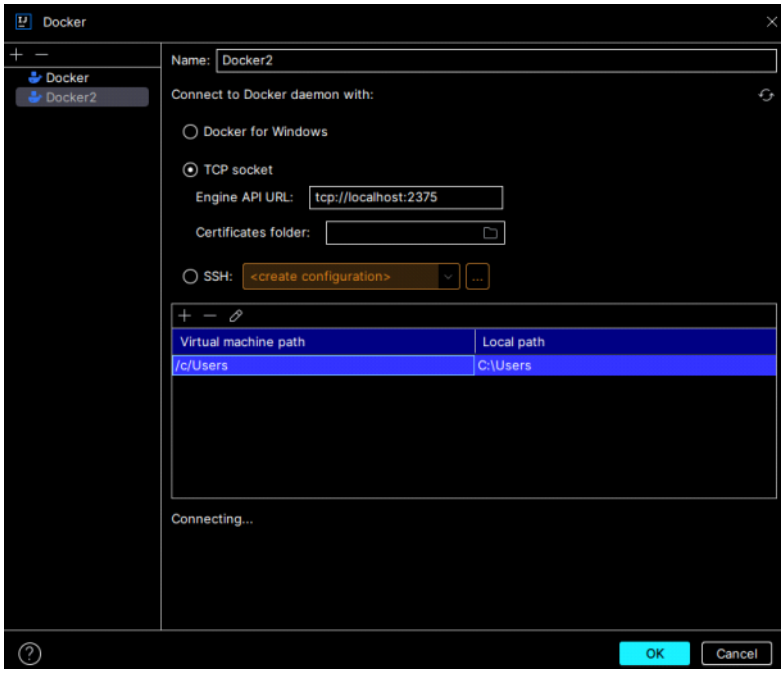
Thursday, April 11, 2024 10:38 AM

1) Alt+8 or Services icon->



Right click-> Docker-> Edit configuration




Docker
Docker
Docker2
Name: Docker2
Connect to Docker daemon with:
Docker for Windows
TCP socket
Engine API URL: tcp://localhost:2375
Certificates folder:
SSH: <create configuration>
Virtual machine path
Local path
/c/Users
C:\Users
Connecting...
OK
Cancel

IntelliJ Idea and JWT token generation

Tuesday, January 30, 2024 11:05 AM

Token generation in springboot project in intellij idea->

Here are no actual values

1) Project name-> Right click-> New-> HTTP request

testfile.http
##Getprojects
GET http://localhost:8080/api/v1/projects?page=0&size=20
Authorization : Bearer {{JWT_TOKEN}}

Environment-
http-client.env.json

```
{
 "dev1": {
      "JWT_TOKEN": "……."
}
}
```

---

2) The method to generate JWT token inside springboot project:-

i) In src/main/resources- http-client.env.json

```
{
 "dev": {
  "Security": {
   "Auth": {
    "dev-config": {
     "Type": "OAuth2",
     "Grant Type": "Authorization Code",
     "Client Credentials": "in body",
     "Client ID": "something",
     "Client Secret": "something",
     "Token URL": "{{azureTokenUrl}}",
     "Auth URL": "{{azureAuthUrl}}",
     "Redirect URL": "http://localhost",
     "Scope": "api://82830216---------"
    }
```

```
    }
   },
   "azureAuthUrl": "https://login.microsoftonline.com/----------",
   "azureTokenUrl": "https://login.microsoftonline.com/----------"
  }
}
```

ii) In src/test/http folder- Create a .http file

```
GET  some link
Authorization: Bearer {{$auth.token("dev-config")}}
```

GitHub copilot

Wednesday, August 2, 2023 10:37 AM

GitHub copilot-
Works with all major programming languages. Plugin is available in VSCode, Jetbrains' IDEs. It offers suggestions while coding. Suggestions are generated based on context of code and libraries in the code. Write accurate comments based on which code will be generated.

IntelliJ and GitHub copilot

Friday, March 1, 2024 12:43 PM

1. Getting started with GitHub Copilot - GitHub Docs

2.

Plugin homepage-> Versions-> selected 3rd one(19th Feb.)-> Downloaded zip -> Plugin settings- Install plugin from disk.

For me, a previous version of github copilot worked.

_____

Meeting notes- IDE

Thursday, February 29, 2024 4:57 PM

Currently backend project provides .editorconfig file which defines how code should be formatted.
However, this file is not respected by IDE and it fallbacks to the default settings.
Happily as long as nobody from development team made changes into their settings code was formatted accordingly - but not compliant to expected definitions.

Why IDE does not use .editorconfig file as expected?

Because it fails to parse it. There is committed unresolved conflict change for the formatting of Java imports.

Why not fix that immediately in scope of one of the pull requests?

Because when did so and formatted code accordingly it was so many changes that nobody would catch from PR and review actual changes from formatting.

We need dedicated PR and align with all developers because it will conflict a lot.

- possible that some adjustments to .editorconfig settings should be made.
- worth to check whether they are compliant with checkstyle and sync them.

IntelliJ Idea Ultimate

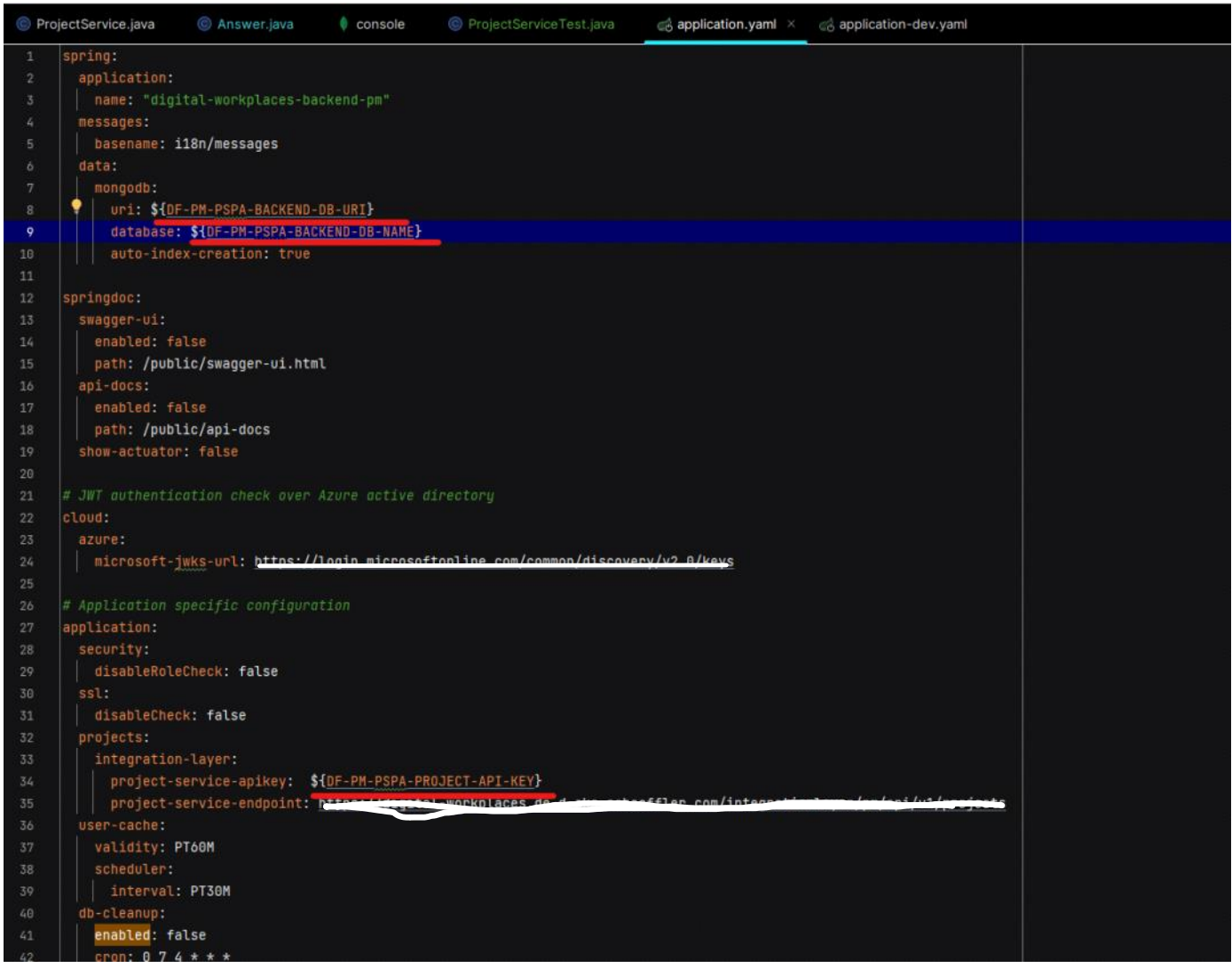
Friday, March 1, 2024 12:49 PM

IntelliJ Idea ultimate is paid version for which I got license activation code from my firm.

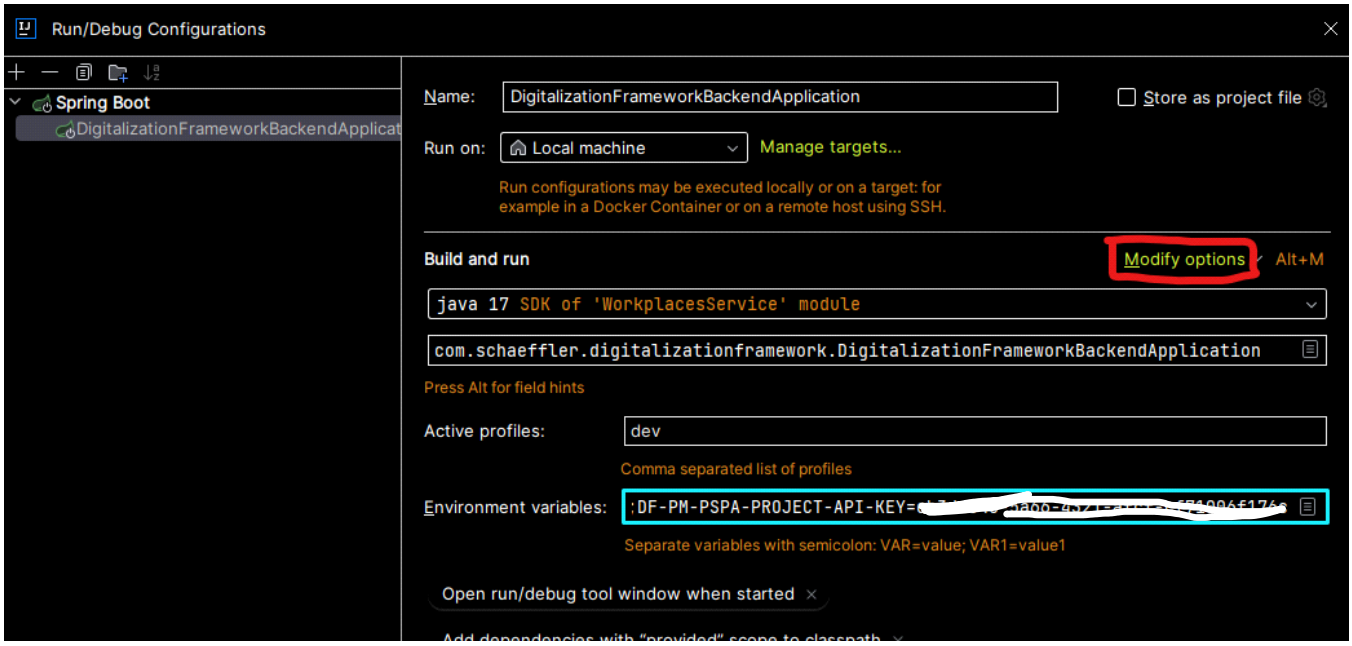
IntelliJ- Environment variables

Monday, January 22, 2024 3:12 PM

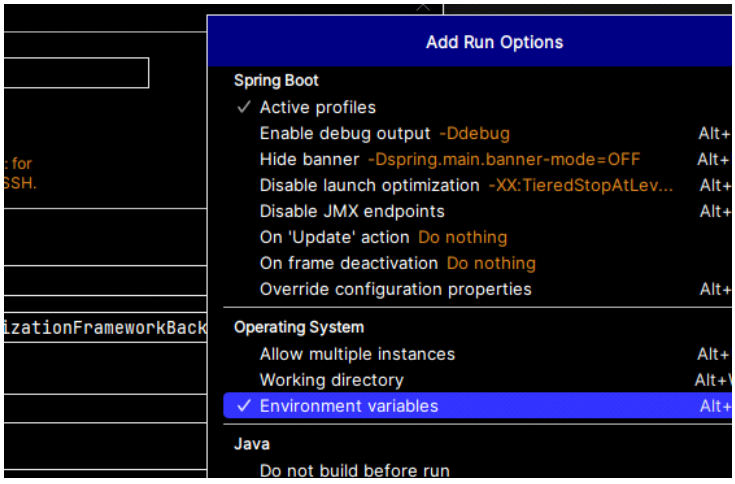
There are variables in application.yaml whose value needs to be added using environment variables from 'Edit configurations' option.

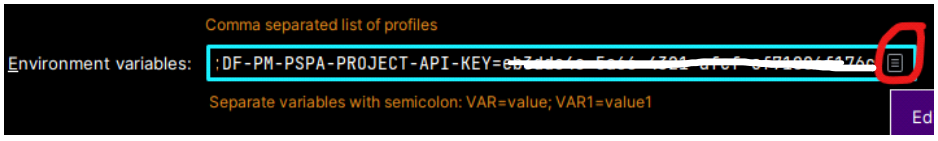


Add values of variables from environment variables.
Click on - Edit Configurations

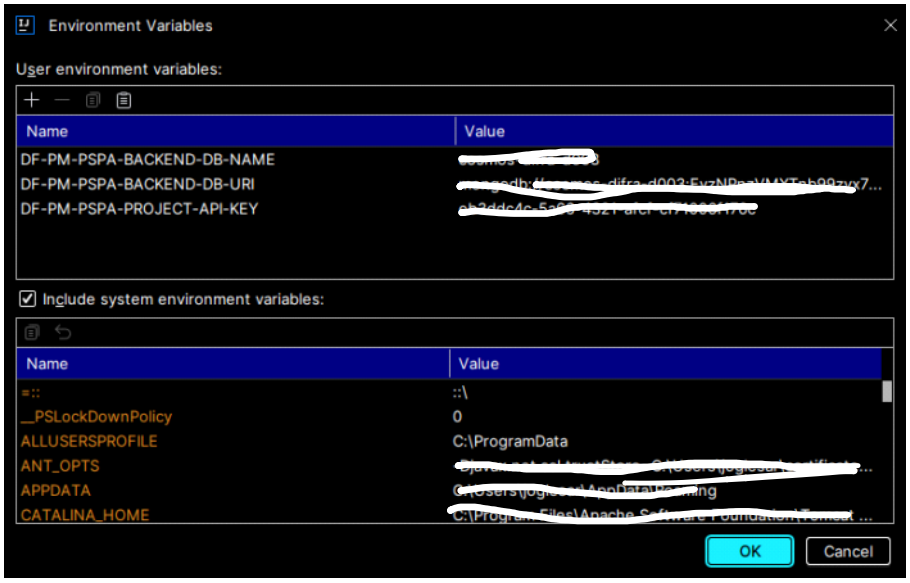


Click- Modify options-> Environment variables.





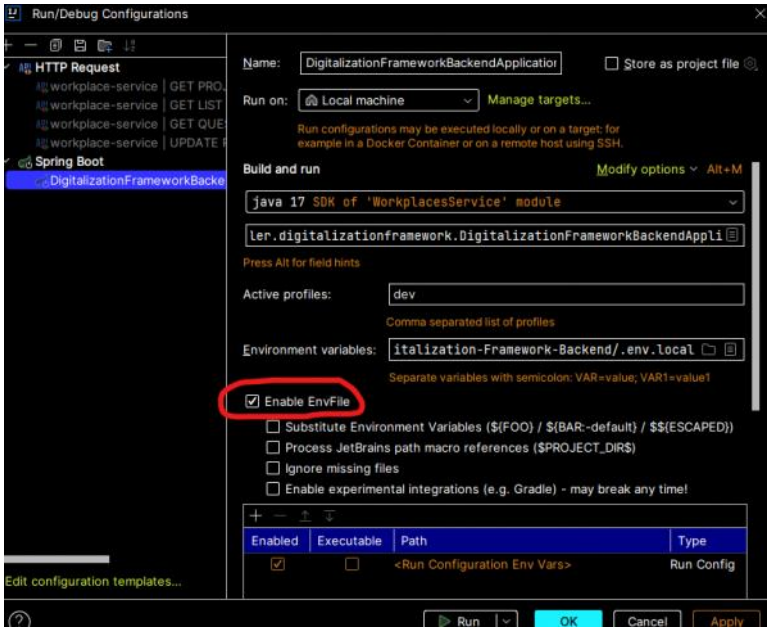
DF-PM-PSPA-BACKEND-DB-NAME =
DF-PM-PSPA-BACKEND-DB-UR I=
DF-PM-PSPA-PROJECT-API-KEY=
I copied above 3 variables and values without any spaces and pasted in Environment variables section.



---

2. Using Env file to add environment variables

Installed a plugin in IntelliJ idea- Envfile
I created a file in my project folder- .env.local
In this file-> I pasted the environment variables with their values sent by colleague.
Click on Enable EnvFile from Run configurations.

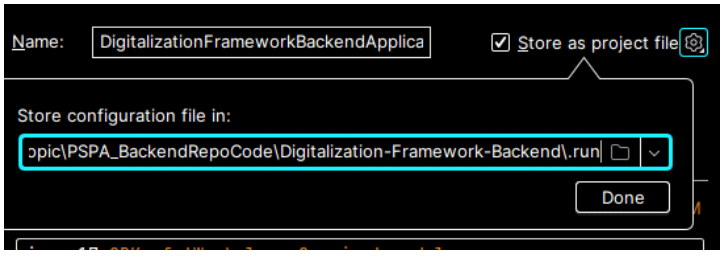


IntelliJ- Using configuration of project

Thursday, March 28, 2024 11:14 AM

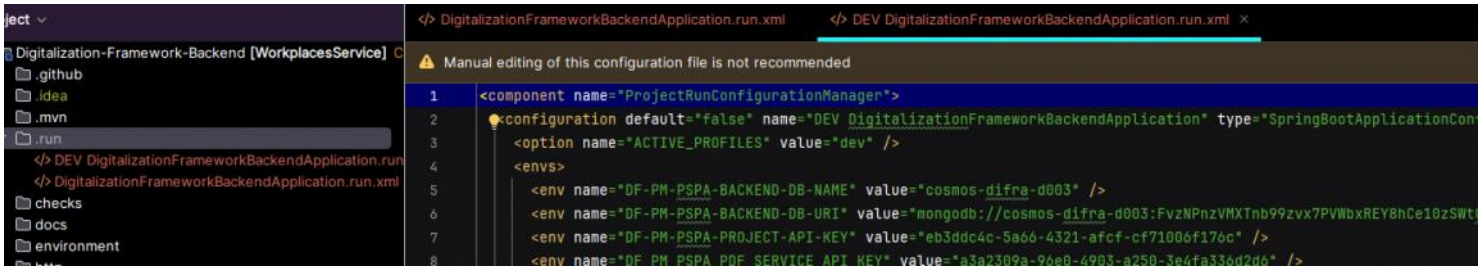
1) I had error in running the project probably due to misconfiguration of environment variables. So, Gabriela shared me her configuration of environment variables.

Edit configurations-> Check the box- Store as project file-> Click on Done for .run



Click Apply and Ok.

You can see .run folder in your project. There'll be already a .xml file present.



In that folder, paste the other .xml file (It was sent to me by Gabriela).

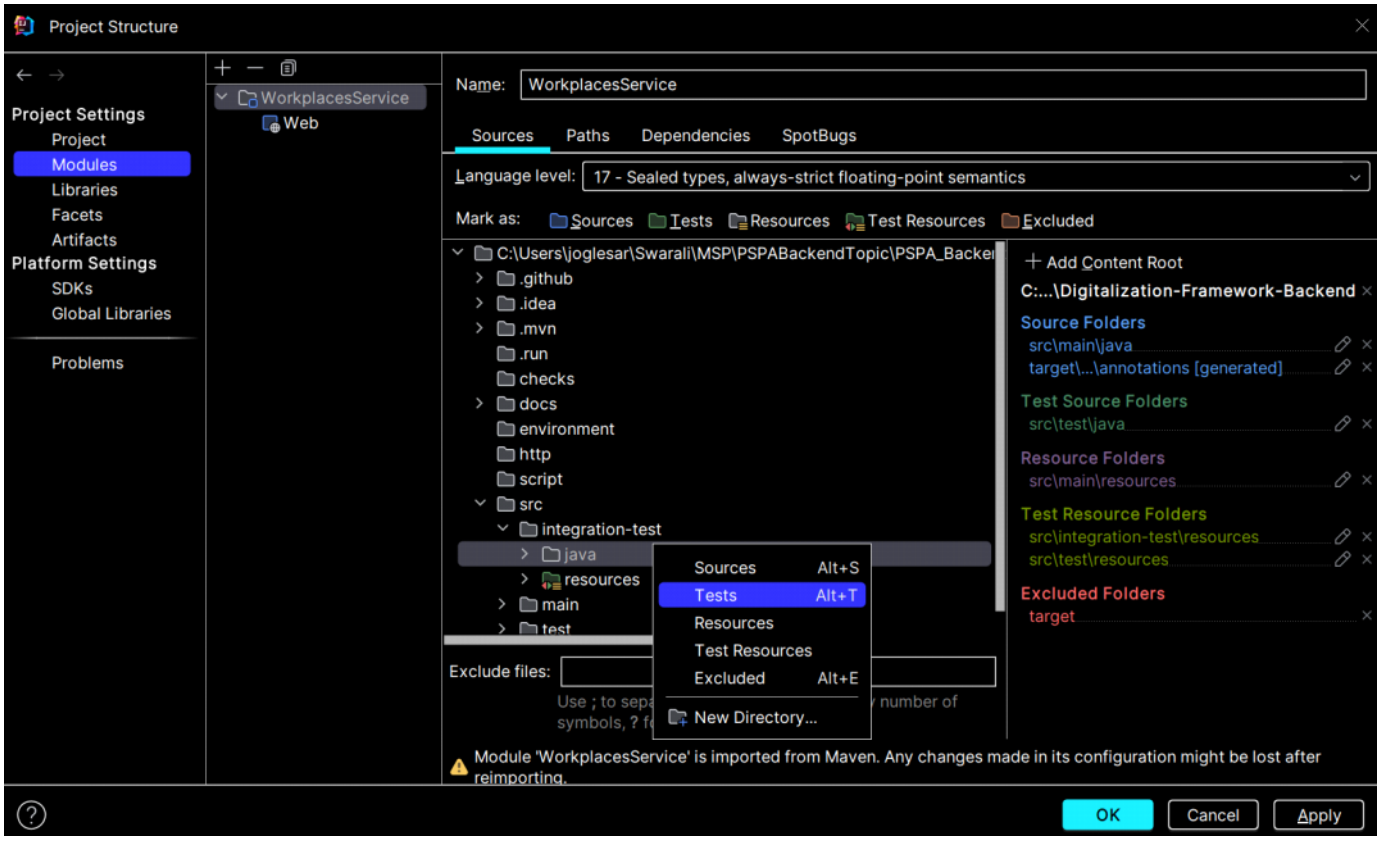
Afterwards, project's execution worked.

IntelliJ- problems and solutions

Thursday, April 4, 2024 11:28 AM

1) Test- play button not visible-

Solution:



File-> project structure-> modules-> java folder-> right click -> select Tests-> ok

---

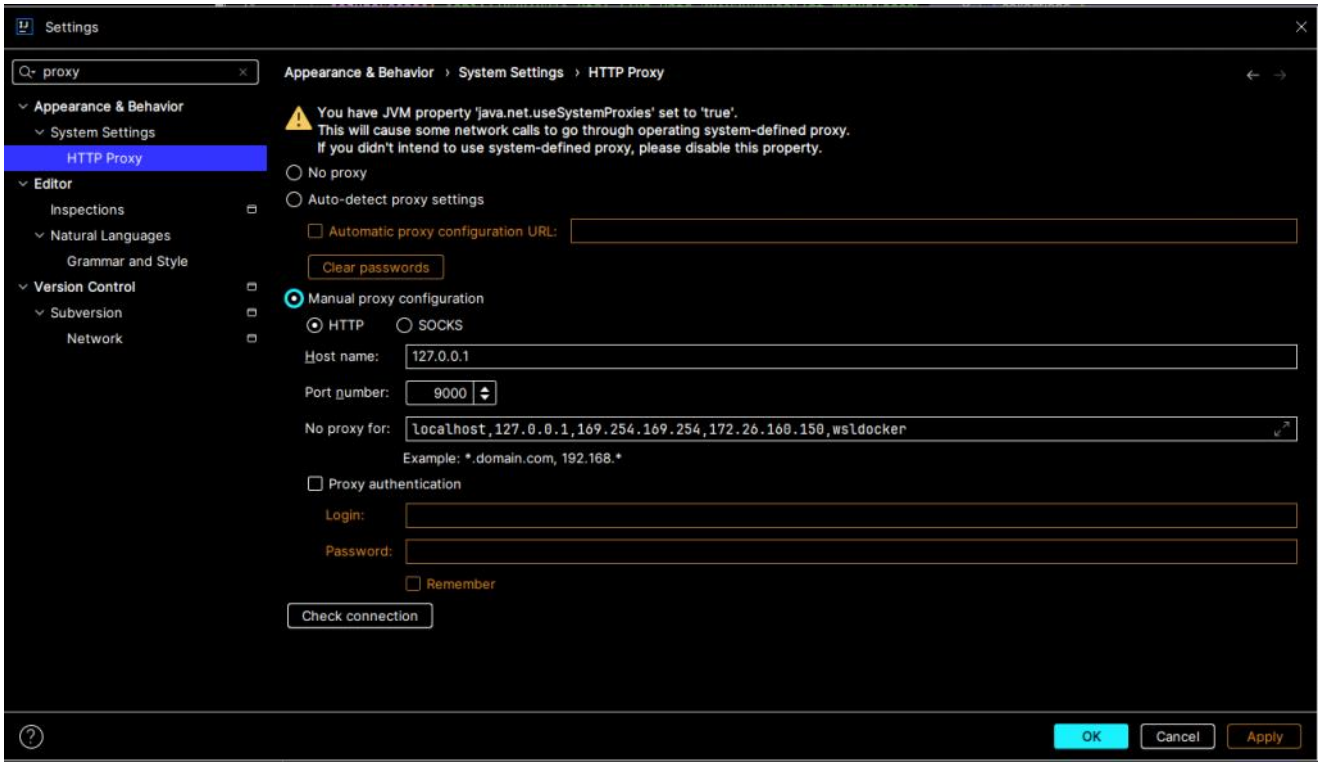
2) On getting build failure, you can try the following options.

Build menu-> Build project, Rebuild project
File menu-> Invalidate caches, Repair IDE

IntelliJ settings for office laptop

Thursday, April 11, 2024 10:46 AM

1) Proxy settings for office laptop



2) File-> Settings-> Search- Server certificates- Make sure that certificates are added.
Refer Schaeffler's confluence page-> JetBrains IntelliJ

IntelliJ IDEA stores data to make working with your code faster and more efficient. Here’s why:

1. **Indexing**: IntelliJ indexes your code to enable features like search, code completion, and navigation. This helps you quickly find classes, methods, and other code elements without having to search through the entire codebase every time.

2. **Caching**: It caches various aspects of your project to speed up operations. For instance, it keeps track of which files are part of your project and the state of your build artifacts. This way, IntelliJ doesn’t have to reload or reprocess everything from scratch each time you open the project.

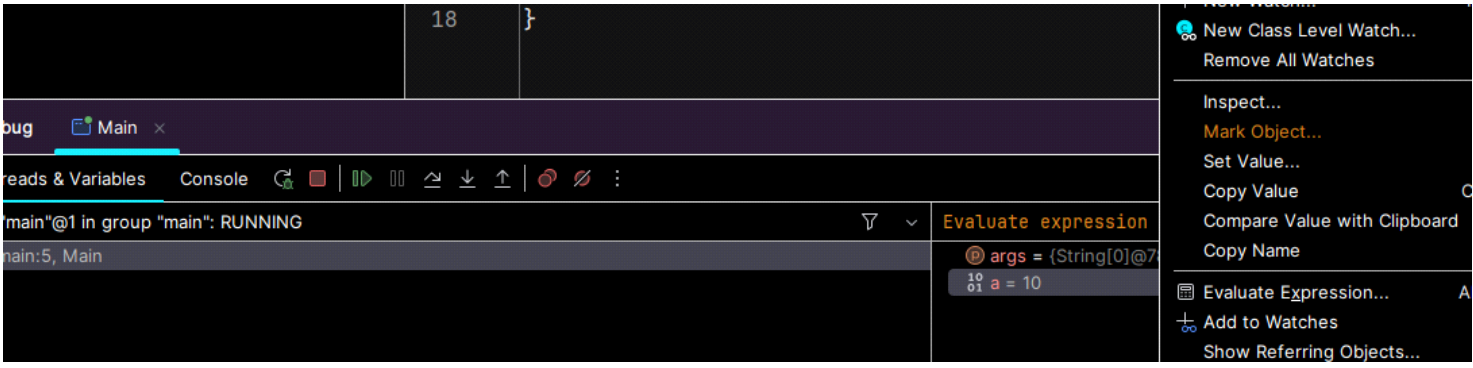
3. **Performance**: Storing and caching this data helps IntelliJ perform tasks more quickly. For example, it can quickly provide suggestions as you type or show code errors in real-time because it already has some information about your code.

Sometimes, though, this cached data can become outdated or corrupted, leading to issues like incorrect error messages or slow performance. When you "Invalidate Caches," you clear out this old data so IntelliJ can rebuild it correctly and get everything back in sync with your actual code.

IntelliJ debugging

Friday, January 17, 2025 6:57 PM

1) Start debugging
Right click on the variable-> Set value



Part1:Java- Office Group Study

Monday, June 17, 2024 6:19 PM

1. 17th June 2024
Topics:- OOP, Access modifier, Encapsulation

Object oriented programming= class based programming
Real world object- has state and behaviour

Local variables- stored in stack memory
Instance variables (Global, static variables)- stored in heap memory

static method cannot call non-static method.
non-static method can call static method.

QUESTION- Can local variable be static?

class name- 1st letter- upper case

protected access modifier- same package access and subclasses in other package access

Encapsulation- i) Binding of variables and methods ii) Data hiding using access modifier

class can't be protected or private.

---

2. 18th June 2024
Topics:- default value of instance variables, Getter and setter methods

If there is instance variable of primitive data type in a class and if any value is not assigned to it-> it'll have default value
For e.g.

```
class A{

	public int x;

	public void print(){
		System.out.println(x); //0
	}
}
```

But local variable of primitive data type if not assigned any value-> Error

String is not primitive data type. String when unassigned, has null.

Getter and setter methods:- To control and protect access to private fields.

IntelliJ Idea- Put cursor inside the class-> Right click-> Generate-> Getters and setters

Naming standard of getter methods for non-boolean data types- getFieldName()
Naming standard of getter methods for boolean data types- isFieldName()

e.g. boolean convertible -> isConvertible()

_____

3. 19th June 2024-
Topics :- Setter, Creating objects

1. Setter- return type void
2. Warning- reassigned
public void setMake(String make){
make = make;
}

Correction:-
this.make= make;

3. **this** - reference name for the object or instance

4. toLowerCase() - method in String

5. We can write validation inside setter.
for e.g. to withdraw the amount in bank account, inside the setter we can check whether the withdrawal amount doesn't exceed the account balance.

6.
Compile time error- Local variable can't be uninitialized.
RuntimeException- NullPointerException

7.
Although phoneNumber would be Long, if you don't write L here, you'll get error that integer is too large.
acc.setPhoneNumber(9999999999);

Account acc = new Account();
acc.setPhoneNumber(9999999999L);

_____

4. 20th June 2024

i. default constructor- no arguments constructor

ii. Constructor- Initializing the object

Constructor- has no return type, has access modifier, name same as class name

Constructor is called while creating the object.

iii. Parameterized constructor

iv. Constructor overloading
Types and order of types of parameters should be different

v. isEmpty() - method in java to check whether string is empty

_____

5. 21st June 2024

i. Constructor chaining:
Constructor explicitly calls another overloaded constructor using this().

```
class Account{

	public Account(){
		this(23, "Swarali");
	}
	public Account(int age, String name){
		System.out.println(age+ " "+ name);
	}
}
```

ii. this() must be the first statement in constructor body.
_____

6. 24th June 2024

i)
Analogies to house:
blueprint of house-- class
house-- object
address of house-- reference

ii)
Both references pointing to same object
House blueHouse = new House("blue");
House anotherHouse = blueHouse;

Dereferencing and referencing
Housse newHouse = new House("red");
anotherHouse = newHouse; //dereferencing anotherHouse and referencing to another object

iii)
new House("red");
This line compiles fine. But, code has no way to access this object.
This line is eligible for garbage collection.

iv)
static variable:
- instance of class share static variable
- Accessed with class name
- Applications- Storing counters, creating and controlling access to shared resource, Generating id

```
class Student{
	private static String collegeName;
	public Student(String collegeName){
		this.collegeName = collegeName;
	}
}
```

```
Student stu1= new Student("PCCOE");
Student stu2 = new Student("COEP");
```

For both of above references stu1, stu2, collegeName will be same as COEP.
Because, collegeName is static variable.

Instance variable: fields, member variables.
Each instance has its own copy of instance variables.
_____

7. 25th June 2024
Topic:- Static methods, Plain Old Java Object

this cannot be used inside static method.

i) static method in a class:
- Called from another static method in same class- directly call with method name
- Called from another static method from another class- call with className.methodName()
- Called from instance method in same class- directly call method name
- Called from instance method in another class- call with className.methodName()

ii) instance method in a class:
- Called from static method in same class-> Create an instance and call from instance
---

Plain Old Java Object- only has instance fields
- database frameworks use it to read data from or write data to database, files or streams.

Java bean- POJO with extra rules

```
public class Main {

    public static void main(String[] args) {

        for (int i = 1; i <= 5; i++) {
            Student s = new Student( id: "S92300" + i,
                    switch (i) {
                        case 1 -> "Mary";
                        case 2 -> "Carol";
                        case 3 -> "Tim";
                        case 4 -> "Harry";
                        case 5 -> "Lisa";
                        default -> "Anonymous";
                    },
                    dateOfBirth: "05/11/1985",
                    classList: "Java Masterclass");
        }
    }
}
```

Object class has toString() method.

We can override it for our class.
toString()- prints current state of object.

default implementation of toString() is a String with class name and memory address of object.

```
@Override
public String toString() {
return rollno + " " + name + " " + city;
}
```

System.out.println(referenceToObject) -> this will call our implementation of toString

---
Annotations- type of metadata, used by compiler or other preprocessing function to get more information about code.

Metadata- additional information about code

_____

8. 26th June 2024

record class- introduced in JDK 14, officially part of java from JDK 16.
In the IDE-> src folder- right click- new java class - record

record class
- fields are private and final
- Accessing the field- fieldName(), no getter method
- No setter method- to protect the data from unintended mutations

public record className(parameterDatatype parameterName, ….){
}

We have to create the object as we normally do and parameterized constructor will also be automatically called.

_____

9. 27th June 2024

sqrt()
public static double sqrt(double a) - neeeds argument of type double

import java.lang.Math;
System.out.println(Math.sqrt(9));

or
import static java.lang.Math.sqrt;
System.out.println(sqrt(9));
System.out.println(Math.sqrt(9));

_____

10. 28th June 2024
Solved coding exercise.

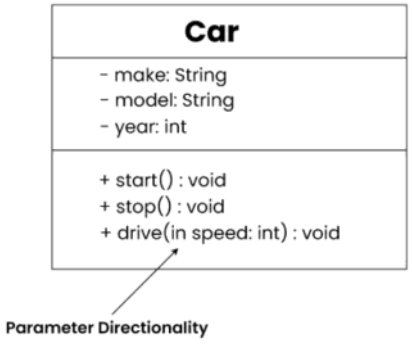
_____
11. 1st July 2024

Inheritance
- child inherits fields, methods from its parent
- **extends** keyword. Only one class can be inherited.

class diagram-

Common visibility notations include:

- + for public (visible to all classes)
- - for private (visible only within the class)
- # for protected (visible to subclasses)
- ~ for package or default visibility (visible to classes in the same package)



super() - can call parent class's constructor
- has to be 1st statement of constructor
- this() and super() are not together

public class Dog extends Animal{ }
This will generate error if:- there is no default constructor in Animal

```
public Dog(){
      super("Mutt", "Big", 50);
}
```

If any method needs superclass's reference, we can pass subclass's reference also.
But not vice versa.

12, 13. 2nd July 2024, 3rd July 2024

Signature of a method= method name, number of parameters, types of parameters
While Overriding methods, signature should be same.

```
public static double pow(double a, double b)
```

Math.pow()

Intellij Idea->
Code-> Override methods

super.move(speed); -> Calling parent class's method move

```
private void bark(){
}
```

Conditional polymorphism- protected

Already present methods in Object class-> hashCode(), equals(), clone(), toString()

String class- overrides equals() and toString()

hashcode is hexadecimal

_____

14.
4th July 2024- Coding Exercise

5th July 2024
Topics:- super, this, method overloading, method overriding

super- used with parent class members
this- used with current class members

this(), super() - used inside constructor
super.methodName()- used to call parent class's method

We can overload static or instance method.

Compile time polymorphism- Method overloading- Compiler decides which method to call
Runtime polymorphism= Dynamic method dispatch

@Override- Compiler can flag an error if overriding is not correct

final method cannot be overridden.

Method Overloading	Method Overriding
Provides functionality to reuse a method name with different parameters.	Used to override a behavior which the class has inherited from the parent class.
Usually in a single class but may also be used in a child class.	**Always in two classes** that have a child-parent or IS-A relationship.
Must have different parameters.	**Must have** the same parameters and same name.
May have different return types.	**Must have** the same return type or covariant return type(child class).
May have different access modifiers(private, protected, public).	**Must NOT** have a lower modifier but may have a higher modifier.
May throw different exceptions.	**Must NOT** throw a new or broader checked exception.

Covariant return type- return type of subclass's overridden method can be a subclass of return type of superclass's method.

```
class Animal {
   Animal giveBirth() {
      return new Animal();
   }
```

```
}

class Dog extends Animal {
   @Override
   Dog giveBirth() {
      return new Dog();
   }
}

public class Main {
   public static void main(String[] args) {
      Animal animal = new Animal();
      Animal animalChild = animal.giveBirth(); // returns an Animal object

      Dog dog = new Dog();
      Dog dogChild = dog.giveBirth(); // returns a Dog object
   }
}
```

Parent reference child object - self learning

```
class Animal {
   public String name;

   public void sound() {
      System.out.println("Some generic animal sound");
   }
}

class Cat extends Animal {
   public String color;

   @Override
   public void sound() {
      System.out.println("Meow");
   }

   public void catSpecificMethod() {
      System.out.println("This is a specific method for Cat");
   }
}

public class Main {
   public static void main(String[] args) {
      Animal animal = new Cat(); // Parent class reference to child class object
      animal.name = "Tom"; // Accessing the name variable from the Animal class
      // animal.color = "Brown"; // This line will not compile because the reference type doesn't have
access to the specific variable of the child class
animal.sound();
   }
}
```

---

15. 8th July 2024

Text block
String textBlock = " " "   " " ";

Escape sequence
\t, \n, \", \\

\u2022

class formatter-
%n - platform specific line separator
%f - float
%.2f
%6d

int age = 25;
String formattedString = String.format("Your age is %d", age);

"Your age is %d".formatted(age);   // pass the arguments that match the specifier
_____

16. 9th July 2024

String
- 60 methods in String
- Categorization of String methods: Inspection methods, Comparing methods, String manipulation methods
-

  length()
  charAt() - character at given index in the string
  isEmpty() - length is zero
  isBlank() - length is zero or only whitespaces are present
  "\t \n" :- this means there are all whitespaces in the string

  indexOf("World")
  indexOf('r')

  String s= "Swarali Joglekar";
  int i = s.lastIndexOf('a'); //index of last occurrence of character

  int j = s.lastIndexOf('a', 2);   //2
  //from index 2 to last index of given string, index of last occurrence of 'a'
  Explanation:
  Step 1- Find the character at index 2
  Step 2- Go from right to left (from beginning of string to given index).
  Step 3- Find the first occurrence of the given character (while going from right to left).
  That will be the last character in the string from index 2.

  If character is not found: -1 is returned

  equals()
  equalsIgnoreCase()

  contentEquals()- can be used to compare String, StringBuilder, StringBuffer

toLowerCase()

startsWith() - checks whether given string starts with given prefix

endsWith() - checks whether given string ends with given suffix

```
 // Given String
    String first = "Geeks for Geeks";
 // Suffix to be matched
    String suffix = "kse";
 // Given String does not end with the above suffix hence return false
   System.out.println(first.endsWith(suffix));
```

---

17. 10th July 2024

string methods:-

str.**substring**(startIndex);
str.**substring(startIndex, endIndex)**; //endIndex is **up to but not including**

String str= **String.join**("/", "25", "11", "1982"); // first argument is delimiter and the later ones are elements.
// 25/11/1982

str= str.**concat**("/");
str.concat("11");

method chaining:-
"25".concat("/").concat("11").concat("/").concat("1982");

Replacing a character in the text:-
str.**replace**(oldCharacter, newCharacter);
str.replace(oldString, newString);

str.replace('/', '-'); // in String str replace all occurrences of / with -
str.replace("2", "00");

str.replaceFirst("/", "-")
str.replaceAll("/", "---")

Multiline string with escape sequence:-
"ABC\n".**repeat**(3);
Output-
ABC
ABC
ABC

"ABC\n".**repeat**(3).**indent**(8)
// ABC 3 times on each line with 8 spaces at the beginning of every line

" ABC\n".**repeat(3).indent(-2)**
//2 spaces in the beginning will be reduced due to -2 in indent

trim() - removes whitespaces in the beginning and end of string
- also removes whitespace that occurs due to \n, \t

strip()- removes whitespaces, escape sequences for white spaces and also unicode characters for white space. (except \u2007 - we tried this)

unicode- has hexadecimal
In Character Map application in windows, if you click on a character you can view its unicode.

---

18. 11th July 2024

StringBuilder class- mutable class

StringBuilder variable- literal cannot be assigned

StringBuilder stringBuilder = new StringBuilder("Hello"+"World");

String
concat()- you have to assign the result to String to see the concatenation
Name = firstName.concat(lastName);

StringBuilder
- since this is mutable class, we can make change in the object.
- .append("World") , append("a".repeat(17))

new StringBuilder(n) - n is capacity of StringBuilder
by default capacity of StringBuilder is 16.

setLength- to truncate
stringBuilder.reverse().setLength(7);

```
StringBuilder replace(int start, int end, String str)
```

I searched on chatbot:
StringBuilder:
- developers have to handle buffer and its capacity
- string manipulation of mutable strings

---

19. 15th July 2024

Inheritance- Is A relationship
Composition- Has A relationship

Composition- way to make combination of classes act like single coherent object.
Composition is more flexible than inheritance.
In inheritance if the if any methods in parent class are added or removed, that changes the child classes.

PersonalComputer - has Motherboard, ComputerCase
Write the references of class Motherboard, ComputeCase in class PersonalComputer

In class PersonalComputer write methods that will call the methods of Motherboard, ComputerCase.

_____

20. 23rd July 2024

Encapsulation:

API= Public contract- tells how to use the class.

- Do not give access to your class's data members directly to external classes. This will prevent bypassing of checks.
- Write checks inside constructor so that right value will be initialized.

Polymorphism:

class Movie
- method watchMovie()

class Adventure extends Movie
- super.watchMovie()

---

this.getClass().getSimpleName()

System.out.printf("%s %n".repeat(3), " abc", "xyz", "def");

_____

PTO- Part 2…

21. 25th July 2024

Polymorphism.. continued

Create a parent class- Movie
It has method- watchMovie()

Create its subclasses- Adventure, Comedy
They have overridden methods- watchMovie()

class Movie has a method that will return the object of one of its subclasses depending on the input given- switch case

```
public static Movie getMovie(String type, String title) {

      return switch (type.toUpperCase().charAt(0)) {
         case 'A' -> new Adventure(title);
         case 'C' -> new Comedy(title);
         case 'S' -> new ScienceFiction(title);
         default -> new Movie(title);
      };
   }
```

then we call watchMovie() method from the reference of the object returned.
Respective watchMovie() method will be called based on the class type of object.

System.out.println(**x>y**? **x-y** : **y-x**);

_____

22. 29th July 2024

Parent class
Child class-

- child class has a method which is not in parent class

In Parent reference and child class object-> On calling that method from parent class's reference-> Compile time error.
So, typecast the child class's object to parent class.

_____

23. 30th July 2024

How to test runtime type?

Object unknownObject= Movie.getMovie();

-> if(unknownObject.getClass.getSimpleName() =="Comedy")
Comedy c = (Comedy) unknownObject;
c.getComedy();

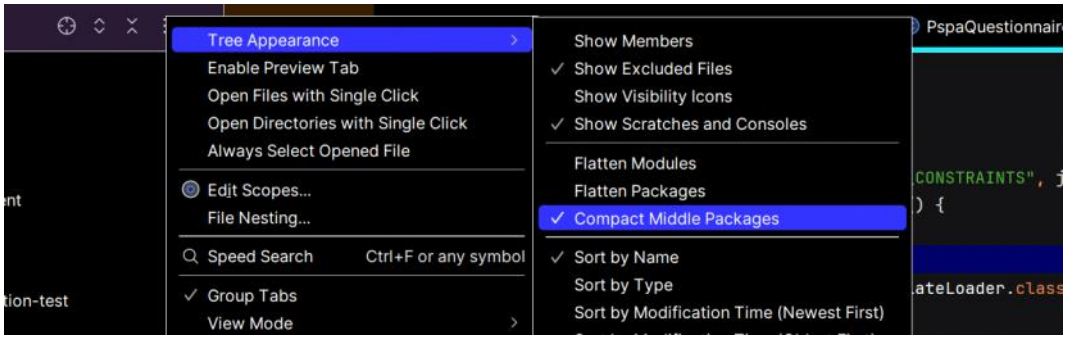
-> if(unknownObject instance of Adventure)
Adventure a = (Adventure) unknownObject;

-> if(unknownObject instance of ScienceFiction syfy)
syfy.watchScienceFiction();

_____

24. 6th September 2024

You can uncheck 'Compact middle packages' in order to see the packages separately.
For example- com.schaeffler will be seen as schaeffler folder inside com folder.



Fully qualified class name can be used instead of import statement.
Example-
java.util.Scanner sc= new java.util.Scanner(System.in);
---
package names should be in lower case.

If you create a new class-> com.schaeffler.digitalization.Item
then, Item class will be created in package com.schaeffler.digitalization

reverse the domain name:
schaeffler.com -> package name should be com.schaeffler

Avoid using default or unnamed package. From default package, classes cannot be used outside.
_____

ProjectLearnings

Thursday, May 9, 2024 6:06 PM

1) @TestContainers

The `@Testcontainers` annotation is used in JUnit 5 tests to indicate that the test class will use Testcontainers. **Testcontainers is a Java library** that provides lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container. It's used for integration testing.

How @TestContainers was used in y project->
`@Testcontainers` annotation is used to signify that the test will use a Docker container for MongoDB as a part of the test environment. This is evident from the usage of `MongoDBContainer` in the code. This allows the tests to run against a real MongoDB instance, providing a more accurate representation of the production environment.

_____

2) @Disabled
"@Disabled" is a JUnit 5 annotation used to disable a test class or a test method.

_____

Higher order functions and lambda(Streams)

Wednesday, December 13, 2023 10:15 AM

Higher order functions accept functions as arguments.

- make code concise
- can perform repetitive tasks on set of data

Functional Interface	Lambda Expression	Example
Consumer	(argument) -> { code }	(n) -> { System.out.println(n); }
Predicate	(argument) -> expression	(n) -> n > 0
Function	(argument) -> expression	(n) -> n * 2
BiConsumer	(arg1, arg2) -> { code }	(a, b) -> { System.out.println(a + b); }
Comparator	(arg1, arg2) -> expression	(a, b) -> a.compareTo(b)
BinaryOperator	(arg1, arg2) -> expression	(a, b) -> a * b

Functional Interface	Example
Consumer	x -> { code }
Predicate	x -> { return boolean }
Function	x -> { return value }
BiConsumer	(x, y) -> { code }
Comparator	(x, y) -> { return int }
BinaryOperator	(x, y) -> { return value (same type) }

Here's a typical loop through an ArrayList:

```
ArrayList<String> list = Arrays.asList("item1", "item2", "item3");
for (String item : list) {
  System.out.println(item);
}
```

This is an example of imperative coding, where you explicitly state every single step. It's functional, but quite verbose. But did you know that Java already provides a higher-order function that has this looping logic already embedded in it? Meet the forEach method.

```
void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
}
```

The `forEach` method asks for a Consumer. The Consumer is an action that declares your intent for each item in the list.

List<String>flowers=Arrays.asList("Daisy","Jasmine","Hibiscus","Rose");
flowers.forEach(flower-> System.out.println(flower)); //for list, forEach() needs **consumer** interface

consumer interface is functional interface. It has accept() method which is abstract.
For looping in list, forEach() method is in **Iterable.java** file.

List<Integer> integers= Arrays.asList(45, 32, 34, 98, 69, 20);

```
//In the internal sorting implementation-
// Initially->parameter1 is 32 and parameter2 is 45.
// 32 is being compared with 45. 32 > 45 => -1 result of compare()
integers.sort((right, left)-> {return right.compareTo(left);} ); //for list, sort() needs Comparator interface
integers.forEach(i-> System.out.println(i)); // for list, forEach() needs Consumer interface
```

```
Map<String,String>map1=new HashMap<>();
map1.put("AtomicHabits","JamesClear");
map1.put("Man's search  for meaning","Victor Frankl");
map1.put("Rich dad poor dad","RobertKiyosaki");

map1.forEach((key,value)->System.out.println(key+"-"+value));//for map, forEach needs BiConsumer interface
```

Stream operations :-

Data source into stream -> processing can be done in pipeline of operations and we can reduce lines of code.

1. Break data source to stream of elements.

Stream is processed as one element at a time.

2. Add filter operation to the pipeline.

filter - receives a stream of elements and returns a filtered stream

Predicate- receives a parameter and returns a boolean

3. Add map operation to the pipeline.
   map - transforms every element in the stream
4. sorted- receives stream as input and returns a sorted stream
5. Add another map operation in the pipeline.
6. Add a forEach operation to the pipeline.

forEach terminates the pipeline.

---

toList()

```
List<String> booksUpperCase= books
            .stream()
            .map(book-> book.toUpperCase())
            .toList();
```

---

reduce() - needs **BinaryOperator**

```
List<Integer> integers= Arrays.asList(45, 32, 12, 89, 54);
      int additionWith10= integers.stream()
            .reduce(10, ((x , y)-> x + y));
```

x is accumulated value of operation and y is current value being processed in stream

---

findFirst() , orElse()

```
List<String> beverages= Arrays.asList(
            "Kashmiri Kahawa",
            "Lassi",
            "Mastani",
            "Ice tea"
      );

 String result = beverages.stream()
            .filter(beverage-> beverage.equals("Ice tea"))
            .findFirst()
            .orElse(null);
```

---

stream for array

```
String[] cities=new String[ ]{"Nurnberg","Bamberg","Munich","Dusseldorf"};
Arrays.stream(cities).forEach(city -> System.out.println( "Hello from "+city ) );
```

---

stream in file->
Lines in file

```
import java.nio.file.Path;

Path path= Paths.get("C:\\Users\\joglesar\\Swarali\\Self-study\\Java\\java projects\\Java-Learning\\src\\com\\swarali\\lambdaAndStreams\\streams\\gpt.txt");
Files.lines(path).forEach(line-> System.out.println(line));
```

lines() method reads lines from file

enum

Wednesday, December 13, 2023 3:17 PM

enum contains limited set of constants.

- When variable is limited to set of values, use enum.
- Used to catch compile-time errors than run-time errors

Car.java

```
package com.swarali.enumTutorial;

public class Car{

   String brand;
   int launchYear;
   public enum TrafficLight {RED, GREEN, YELLOW};

   public Car(String brand, int launchYear) {
      this.brand = brand;
      this.launchYear = launchYear;
   }

//    public void drive(String trafficLight){
//       if(!trafficLight.equals("RED") && !trafficLight.equals("GREEN") && !
trafficLight.equals("YELLOW")){
//       throw new IllegalArgumentException("light should be red or green or yellow");
//       }
//       switch(trafficLight){
//          case "RED":
//             System.out.println("Stop");
//          case "GREEN":
//             System.out.println("Go");
//          case "YELLOW":
//             System.out.println("Slow down");
//       }
//    }

   public void drive(TrafficLight trafficLight){
      if(trafficLight == null) throw new IllegalArgumentException("light should be red or
green or yellow");

      switch(trafficLight){
         case RED:
            System.out.println("Stop");
            break;

         case GREEN:
            System.out.println("Go");
            break;

         case YELLOW:
            System.out.println("Slow down");
            break;
      }

   }

}
```

Main.java

```
package com.swarali.enumTutorial;
import static com.swarali.enumTutorial.Car.TrafficLight;

public class Main {
   public static void main(String[] args) {
      Car car1= new Car("VW", 2001);

      // car1.drive("Magenta"); //I'll get exception from Car.java

      // car1.drive(Car.TrafficLight.YELLOW);

      car1.drive(TrafficLight.YELLOW);
   }
}
```