**INTRODUCTION**

**History of Java**

* Oak (1995-96).
* Developed by James gosling in sun microsystem.
* java is object-oriented programming language.

**Editions**

* java Standard Edition (SE) 🡪Develop standalone application.
* java Micro Edition (ME). 🡪Small size device ex: calculator.
* java Enterprise Edition (EE). 🡪 Organization and business purpose.

**Application**

* Stand-alone application.
* Web Application.

**Why java so popular?**

* java is very easy to learn.
* java has 1000(lot of methods) methods which is difficult to remember. (either it can be taken as advantage or disadvantage).
* java is platform independent (once it is compiled it can work in any of the operating system). i.e. write once use 'N' number of times.

**Development of any of the application we should follow the following steps?**

* First, we write the source code present in source file.
* To make that source file as java source file then we have to save with an extension .java i.e.(source.java).
* Before .java file executes file must be complied i.e. java compiler checks the syntax(rules). java complier checks the syntactical error.
* Command for compiling .java file we use (javac src.java).
* I/P to the java compiler was .java and O/P will be .class
* To execute that .class file we have command (java classname). Whole java works on the concept of virtual (imagination, not real) machine. i.e. JVM (java virtual machine).
* JRE is an Application, it enables to create JVM, in that machine we are able run .class file
* .class file can be run in any of the Operating system that have JRE.
* Inside JVM we have JIT (Just in time compiler, which convert byte codes to binary codes).
* .class file contain byte code (It is a kind a of language neither understand by humans nor system). JVM can understand the byte code.
* JVM acts as an interpreter.
* Byte code is reason for platform independency.
* JIT, it converts byte code into machine understandable language i.e. binary language.

**Working flow of java Program**

JRE

**.**class (ready for execution)

Source.class

JVM

Source.java

JIT

**Java Compiler**

checks syntax

if (syntax correct) {

s.o. p ("generate .class file");}

else {

s.o.p("throws a compile time error");}

**Java program**

**Two parts:**

1. Declaration
2. Definition

**Writing a program, we use either:**

1. class.

public class class name

{

// statements (body, block, definition)}

1. interface.

**Basic program**

If we want to make changes in the .java file again we have to compile it, then only it reflects the changes we made in the .java file

public class HelloJava {

public static void main (String [] args) {

System.out.println("Hello Java");

System.out.println("Hello World"); }}

**Why java is called as Object oriented programming language?**

1. Objects are elements of a program that has some DATA, which is also known as STATES.
2. Objects also has BEHAVIORS which means they can perform certain operation. Those behaviour in java is called as METHODS**.**

**Keywords**

1. Keywords are the words which has pre-defined meaning inside a programming language.
2. Java is case sensitive language.
3. 50 keywords in java. But we have 3 literals (True, False and null) they are also called as keywords. Therefore, we have 53 keywords.

**Identifiers**

* Identifiers are the name given to the variables, class and methods. (Name given to components/Elements of a program).
* RULES to follow when we are creating an identifier:
* 1) It should start with alphabets or a letter.
* It can't start from a digit.
* After the first character it can starts from a digit. Hence it is Alphanumeric.
* Identifiers are also case sensitive.
* Keywords cannot be used as identifiers.
* \_ and $ are the only two special characters can be allowed.
* spaces are not allowed between identifiers when we have two or more identifiers.

**Note**

* If a class is public then you should save that file with same name as class name.
* If we want to execute/Run the java program JRE is enough.
* If we want to developed an application JDK is needed.
* Java is a strongly typed language.
* cmd: (javap classname): we can see what are all the methods present in program(.java).
* Java coding convention (check in google).

**Variables**

* It is piece of memory that has certain name, so we can store certain DATA in a variable.
* Named memory location to store the DATA.
* Every variable should have data type.
* variables can be re-initialized.
* From java 9 var as become a keyword.
* Declaration.
* Initialization
* Utilization.

**Example 1:**

int a; //Declaration.

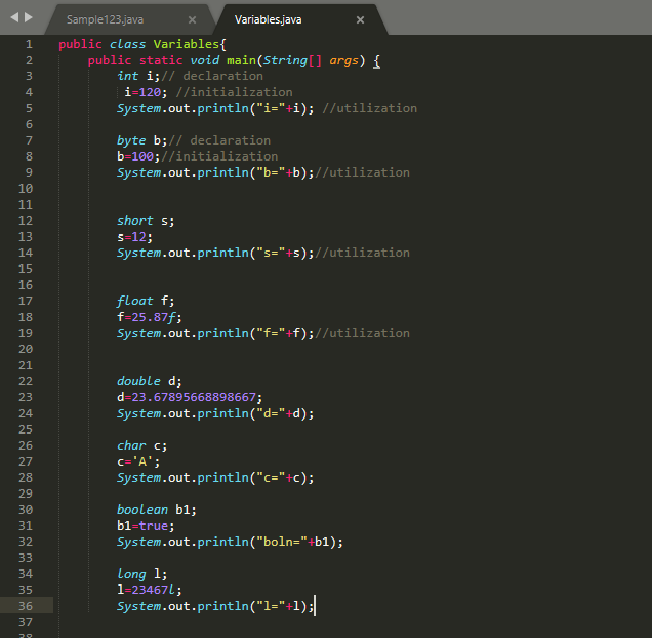
a=10; //Initialization

s.o.p(a); //Utilization.

**Example 2:**

byte b=12;//Declaration and Initialization in same line

s.o.p(b);

****

**Data Types**

* Data type is nothing but it is a type of data which specifies what type of data is a variable is holding.
* Data types are classified into two types namely:

1. Primitive Data type.
2. Reference Data type.

**Primitive Data type:**

A data type which is defined by the language is called as Primitive Data type. Java has 8 primitive data types (out of 8 data types 6 were used for numbers).

1. **byte data type:**

* It can contain numeric value.
* size: 8 bits (1 byte).
* range: -128 to 127.

1. **short data type:**

* It can contain numeric value.
* size: 2 bytes.
* range: -32768 to 32767

1. **integer data type:**

* It can contain numeric value.
* size: 4 bytes.
* range: 2 million approx.
* example: int a=10;

1. **long data type:**

* It can contain numeric value.
* size: 8 bytes.
* range: 9 billion approx.

1. **double data type:**

* It can contain numeric value with decimal point value.
* Decimal point of double can hold more than 6 digits (max:16).
* example: double d=1.123456789
* size: 8 bytes.

1. **float data type:**

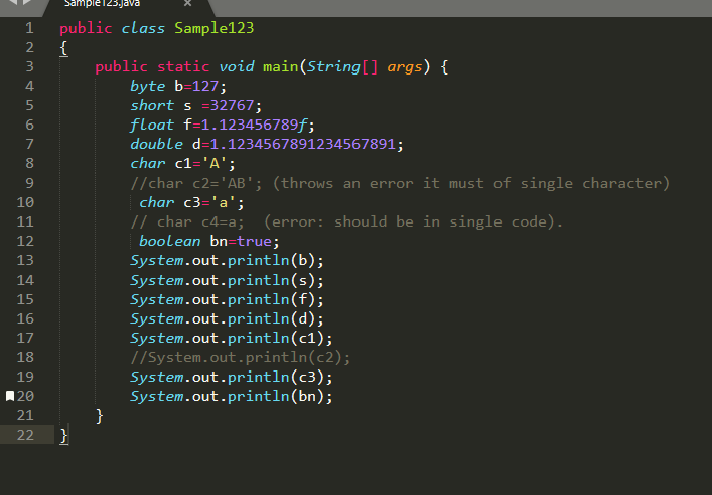
* It can contain numeric value with decimal point value.
* decimal point of float can hold 6 or 7 digits.
* example: float f=1.234567f
* size: 4 bytes.

1. **character data type(literals):**

* It can take only single character.
* It should be written in single quotes.
* example: char c='A';

1. **Boolean data type(literals):**

* It can take the value as true or false.
* Its size is not defined.

****

**Why do we need variables?**

* To perform some operation.
* That Operation is performed by some Operators.

**Operators**

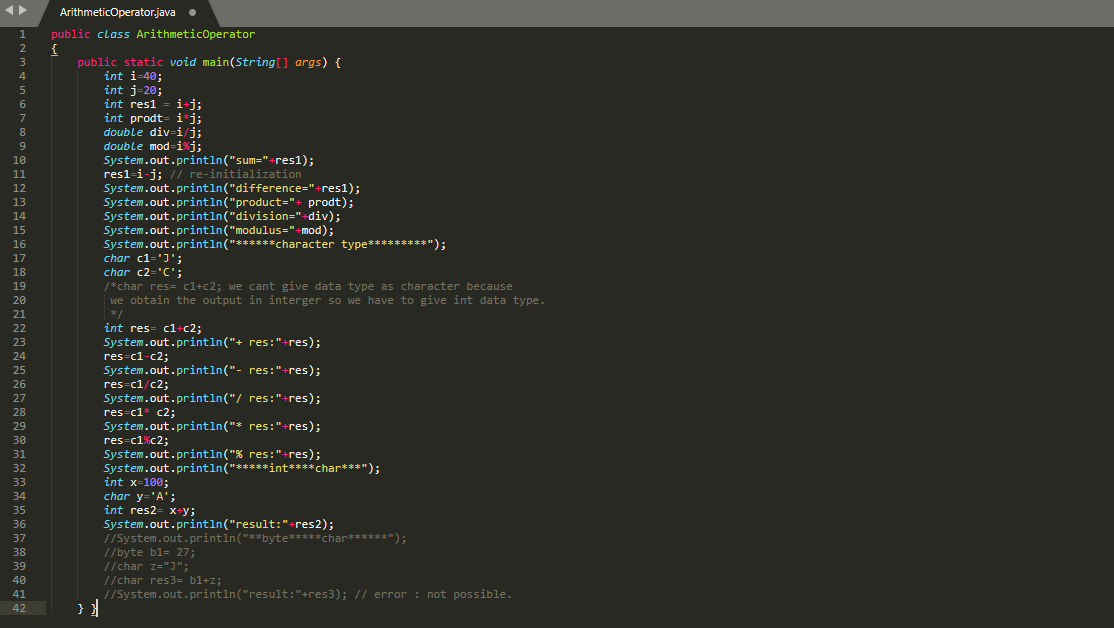
* An Operator is a special symbol or keyword that is used to designate the mathematical operation or some other type of Operation.
* These Operations can be performed on one or more than one values called as OPERANDS.

**Types of operators**

1. Arithmetic operator
2. Assignment operator
3. Increment and decrement operator
4. Relational operators or comparisional operator
5. Logical operator
6. Bitwise operator

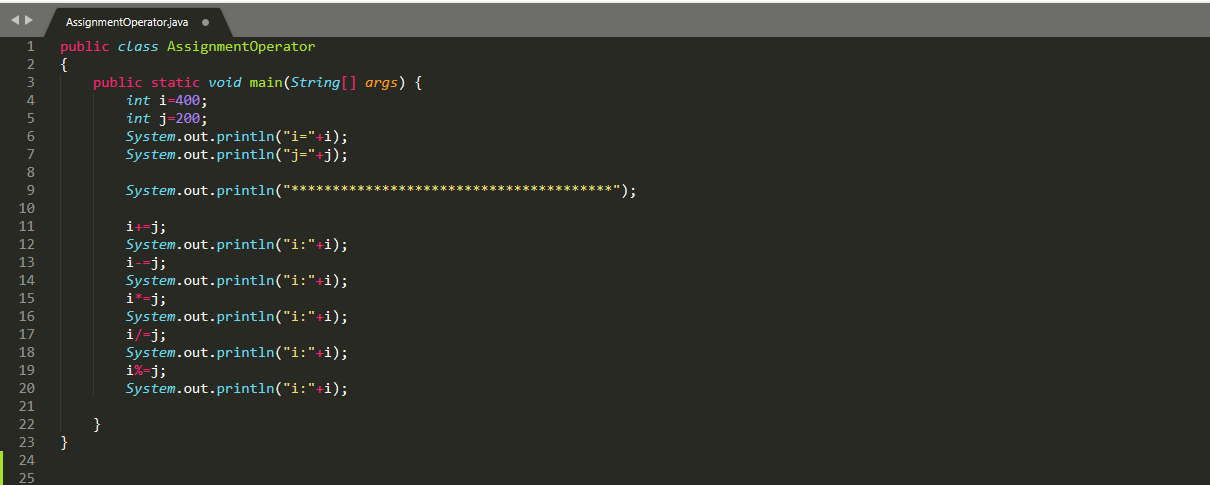
**Arithmetic operator:**

* +, -, \*, /, %
* + and - are Additive arithmetic operator.
* \*, /, % are called multiplicative Operator.
* Example: 4%2 ---> 4/2 ---> 4/1 \* 1/2 (finally we land into Multiplication Operator).



**Assignment operator:**

* =, +=, -=, \*=, /= , % =
* It is also called as Compound Operator.



**Increment and decrement operator:**

* INCREMENT Operator (++) increment the value by one unit.
* DECREMENT Operator (--) decrement the value by one unit.
* These are also called as UNARY OPERATOR.
* When we use the operator before the operand then it is called as:
* pre-increment operator (working flow of the operator).

1. Increment.
2. Substitute.
3. Utilize.

* pre-decrement operator.
* When we use the operator after the operand then it is called as:
* post-increment operator (working flow of the operator).

1. Substitute.
2. Utilize.
3. Increment.

* post-decrement operator.

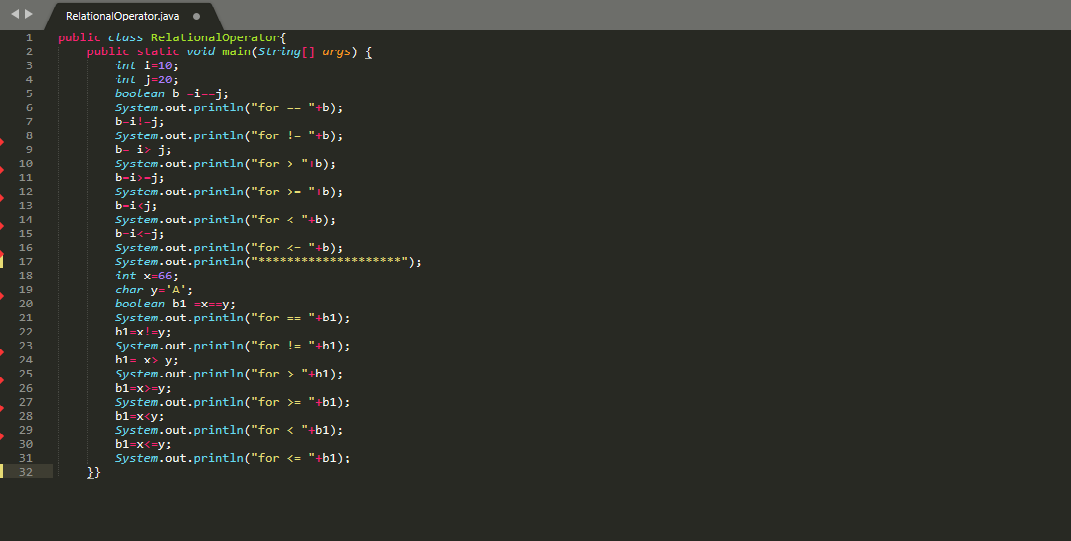


**Relational operators or comparisional operator:**

* They are used to compare the two operands.
* Types of Relational operator:

1. = = It returns true when both side of the equation else false.
2. ! = It returns true When both side of the eqn is not true.
3. < Returns true if the left side of eqn is less than right-side eqn.
4. > Returns true if the left side of eqn is greater than right-side eqn.
5. < = Returns true if the left side of eqn is less than and equal to right-side eqn.
6. > = Returns true if the left side of eqn is greater and equal to than right-side eqn.

* The return type of Relational operator is BOOLEAN.

****

**Logical operator:**

* Whenever we need to check two or more operand at same time we go for LOGICAL OPERATOR.
* Return type of this operator is BOOLEAN.
* Not operator is a unary operator.
* Types:
* NOT (!): It returns false if the RHS is true.
* AND (&&):

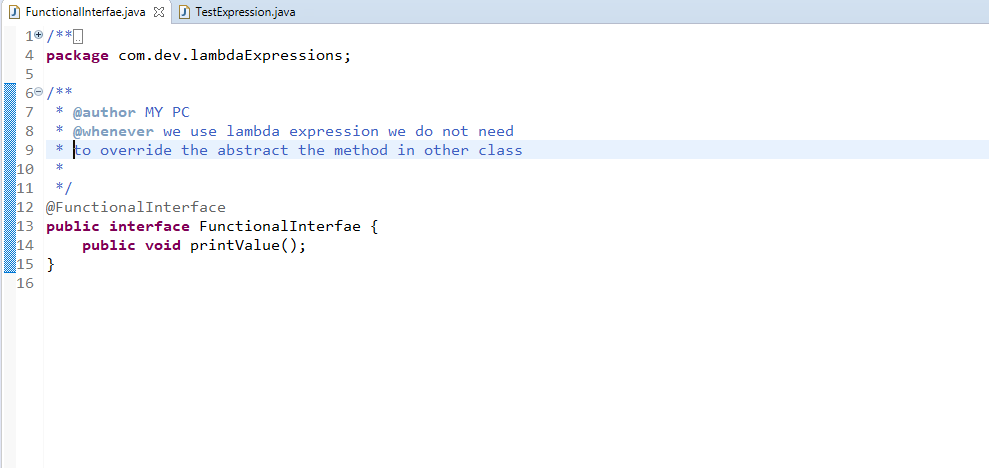
1. It is called as conditional AND or Logical AND operator.
2. Both conditions must be satisfied.
3. If the first condition is false it will not check the condition.
4. OR (||): a) Any one of the conditions must be true.

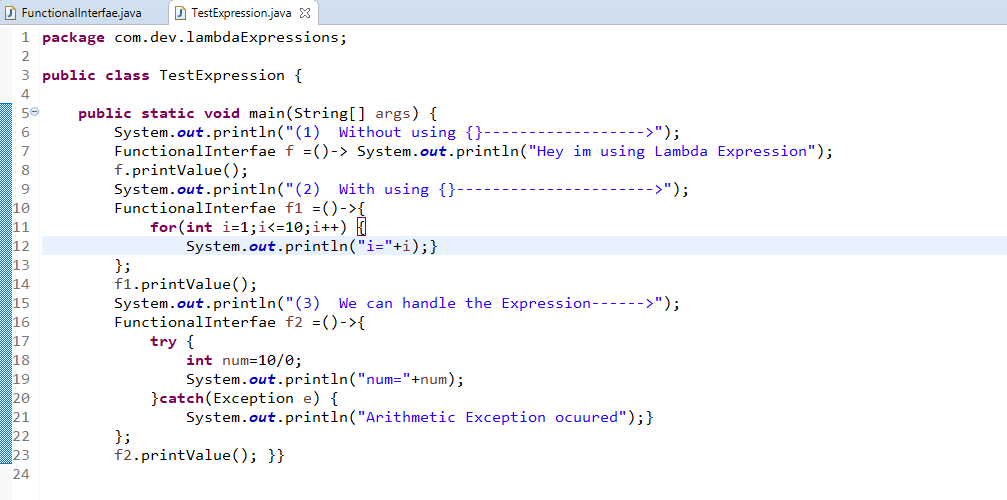
**Java Lambda Expression (java 8)**

* The lambda expression is used to provide the implementation of a functional interface.
* In case of lambda expression, we don’t need to define the method again for providing the implementation.
* **Functional Interface:** only one abstract method, java provides @FunctionalInterface, which is used to declare an interface as functional interface.

**Why do we use Lambda Expression?**

* To provide the implementation of functional interface.
* Less coding

****

****

**Threads**

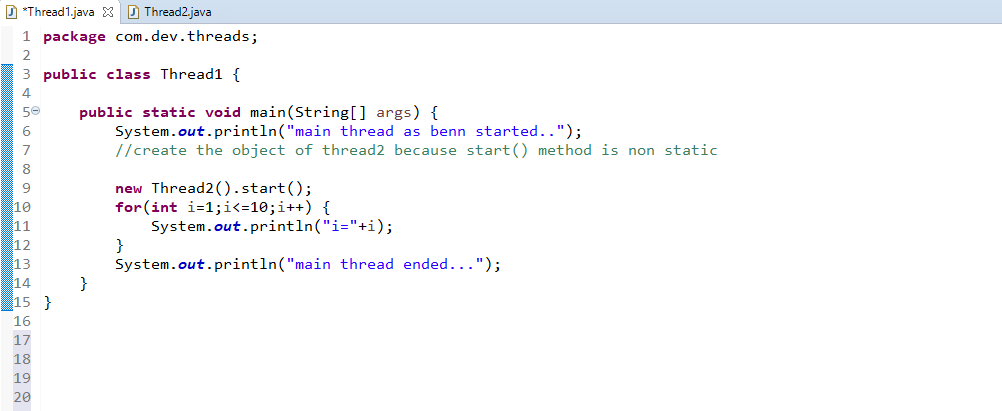
* A thread is a single sequence of executable code within a larger program.
* All the programs so far have used just on thread – the main thread that starts automatically when you run the program.
* But java lets you create programs that start additional threads to perform specific task.

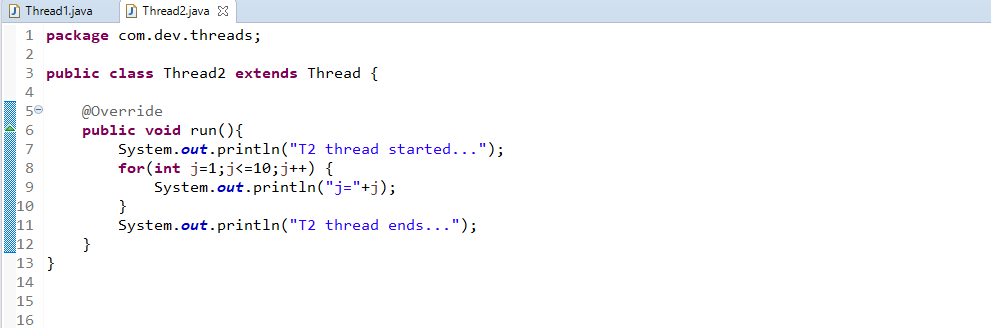
**Multi-Threading**

**Thread Relation**

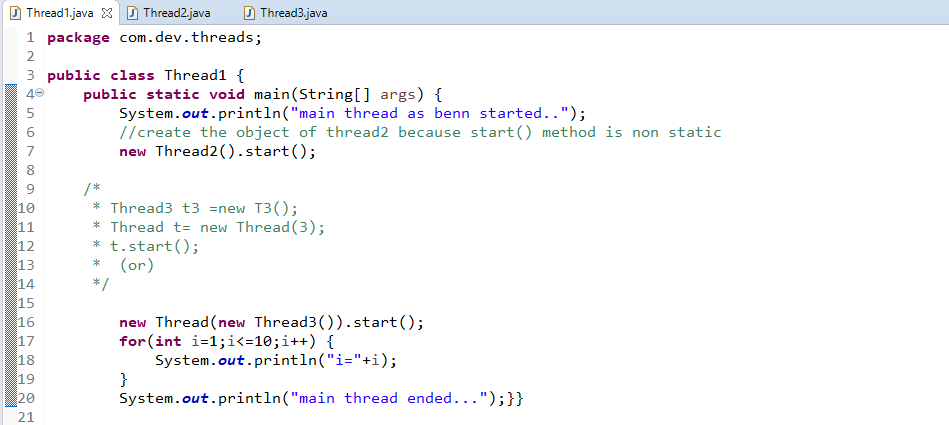
In java threads can be created in two ways:

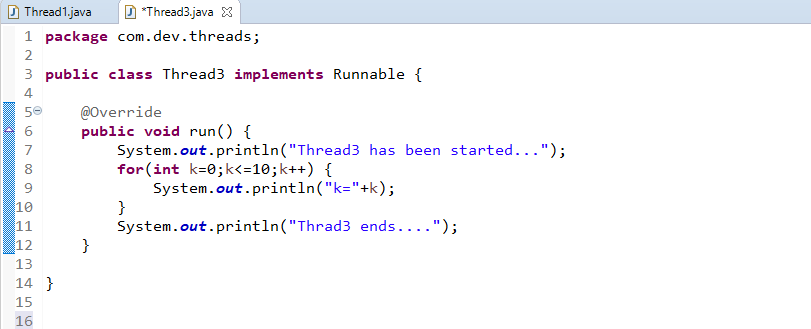
* **By extending the thread class.**





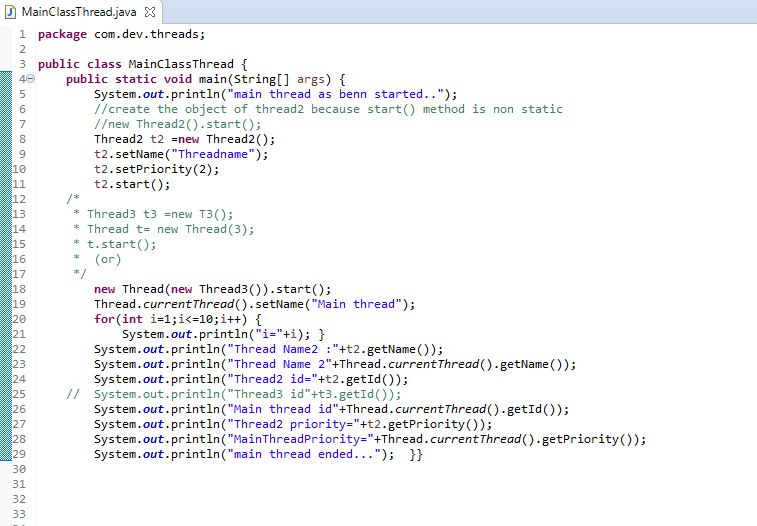
* We create a class that extends the java.lang.Thread class. This class overrides the run ( ) method available in the Thread class.
* **By implementing Runnable Interface.**





* We create a new class which implements java.lang.Runnable interface and override run ( ) method. Then we instantiate a thread object and call the start () method on this object.
* Every thread will have three important properties:

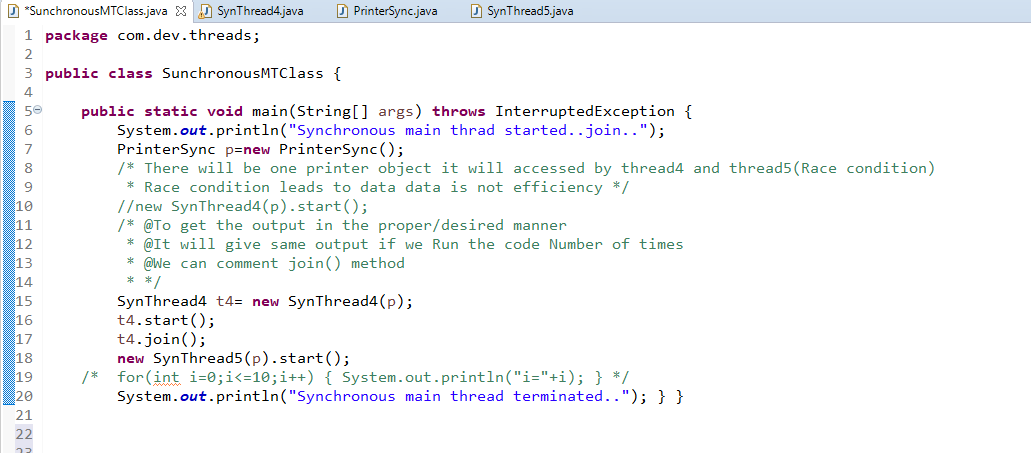
1. Thread Name
2. Thread Id
3. Thread Priority

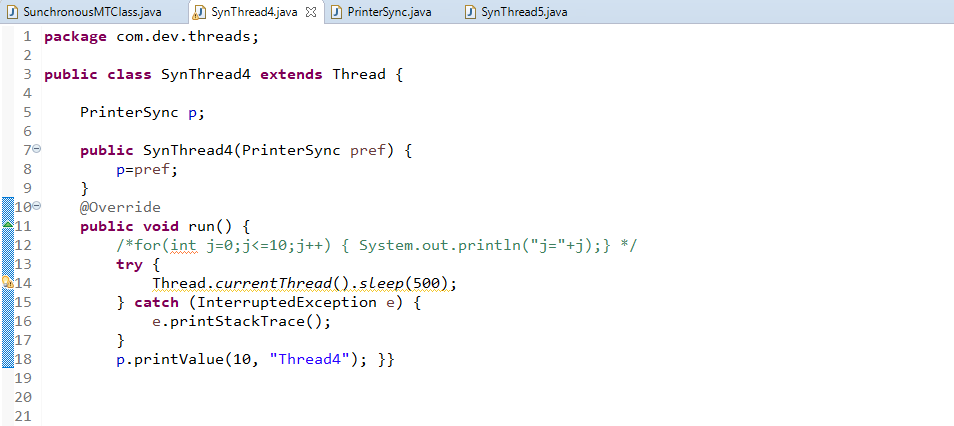


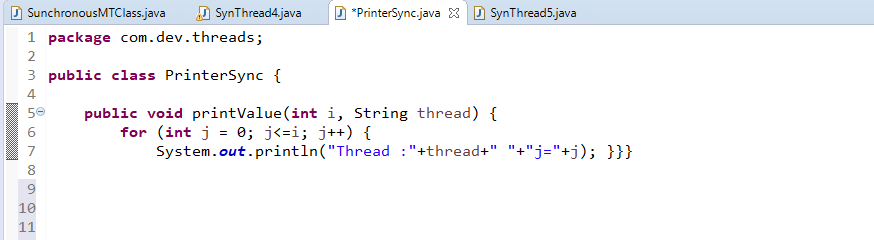
* **In order make data efficient we go for:**

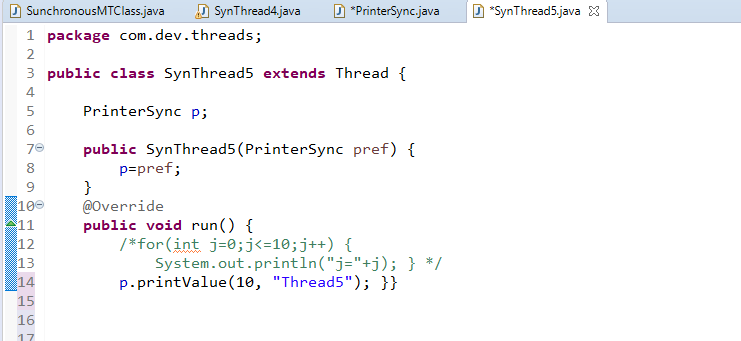
1. join () and sleep ()
2. make method as synchronized

* **join () and sleep ()**

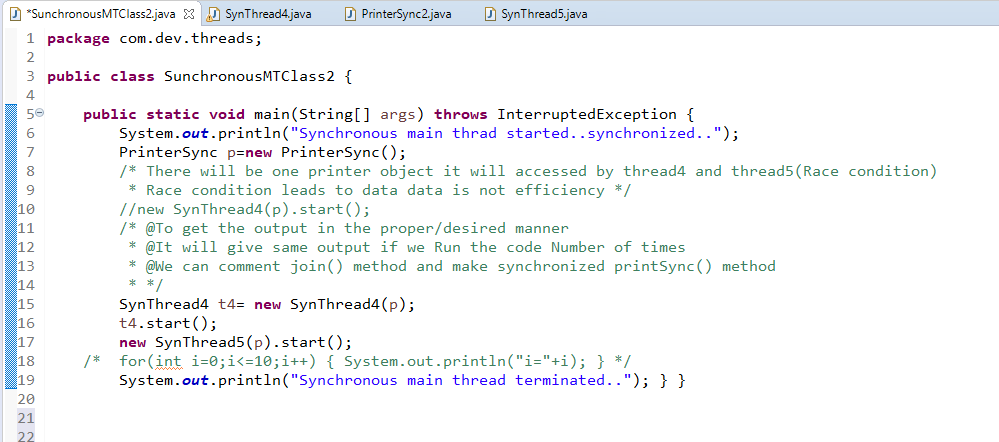
****

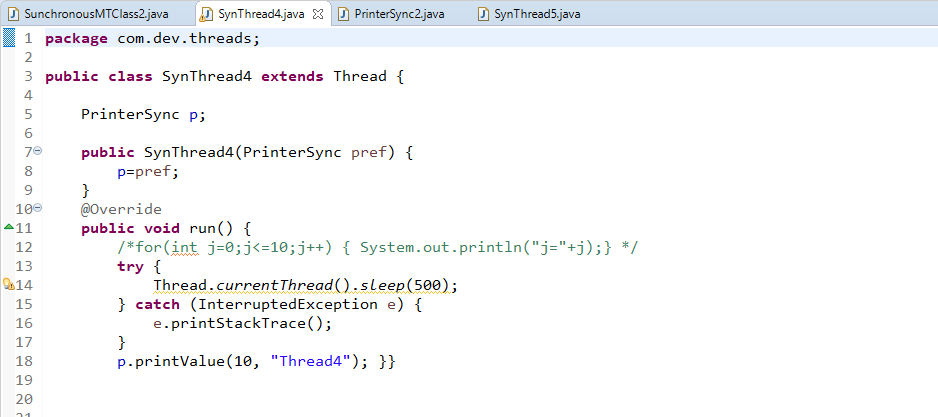
****

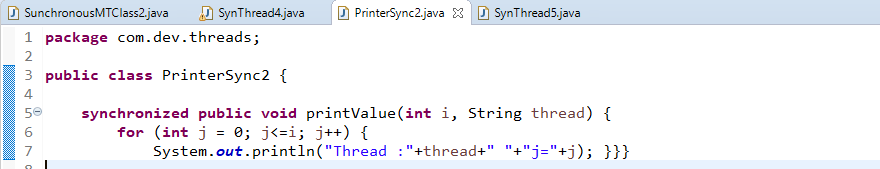
****

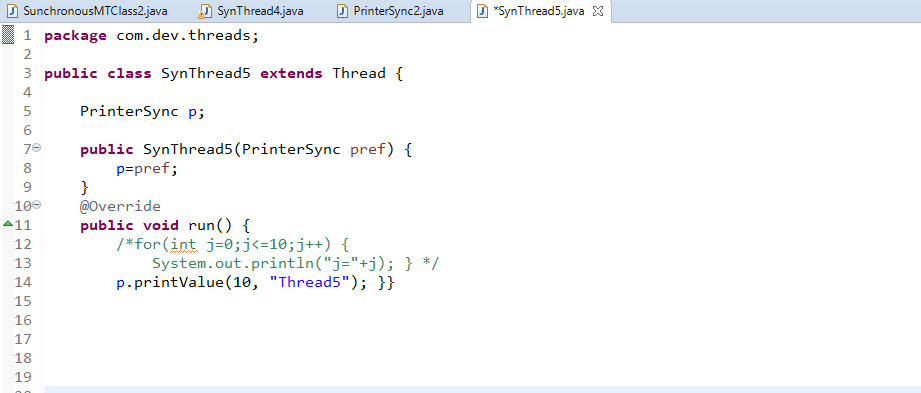
****

* **make method as synchronized:**

****

****

****

****

**Thread Life Cycle**

* The start method creates the system, necessary to run the thread, schedules the thread to run, and calls the thread’s run method.
* A thread becomes “Not Runnable “when one of these events occur:

1. **New:** A thread that has not yet started is in this state.
2. **Runnable:** A thread executing in the java virtual machine is in this state {start ()}
3. **Blocked:** A thread that is blocked waiting for a monitor locks in this state {yield (), sleep (), join ()}
4. **Waiting:** A thread that is waiting indefinitely for another thread to perform a particular action is in this state {wait (), notify (), notifyAll ()}
5. **Terminated:** A thread that has exited is in this state {stop ()}

**Race condition**

* One resource is used by two or more thread.

**Synchronization**

* In many cases concurrently running threads share data and two threads try do operation on the same variables at same time. This often results in corrupt data as two threads try to operate on the same data.
* A popular solution is to provide some kind of lock primitive, only one thread can acquire a particular lock at any particular time. This can be achieved by using a keyword synchronized.
* By using the synchronized only one thread can access the method at a time and a second call will be blocked until the first call returns or wait () is called inside the synchronized method.

**Dead lock**

* Deadlock in java is a programming situation where two or more threads are blocked forever.
* To over come thread deadlock using ITC

**Questions**

1. Difference between run () and start () method?

|  |  |
| --- | --- |
| run () | start () |
| run () method execute when calling to start the method. i.e. just method calling only | start () method cannot call the same thread object more than one time. Then java. lang.IllegalThreadStateException pop up |
| run () invoked directly as normal method | start () internally calls run () method |

1. Difference between sleep () and wait () method?

|  |  |
| --- | --- |
| sleep () | wait () |
| sleep () method belongs to thread class. | wait () method belongs to object class. |
| There is no need to call sleep () from synchronized context. | wait () should be called only from synchronized context. |
| Sleep () does not release the lock on an object during synchronization. | wait () method releases lock during synchronization. |
| Sleep () method execution completes when a thread interrupt. | Wait () method is interrupted by calling notify () or notifyAll () method. |

**Regular Expression (RegEx)**

* Defines a pattern for String (only being used with String).
* It is used to search, edit or manipulate text.
* Java.util.RegularExpression
* **Pattern:** Pattern object is the compiled version of the regular expression. public static method compile () to create the pattern object by passing regular expression argument.
* **Matcher:** Make use of pattern ref and invokes matcher ().