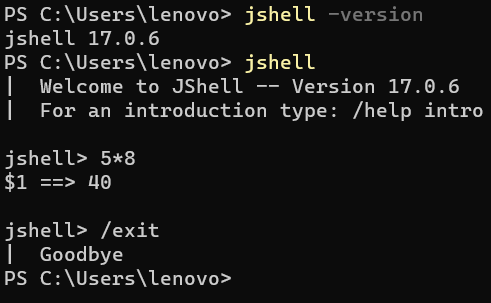
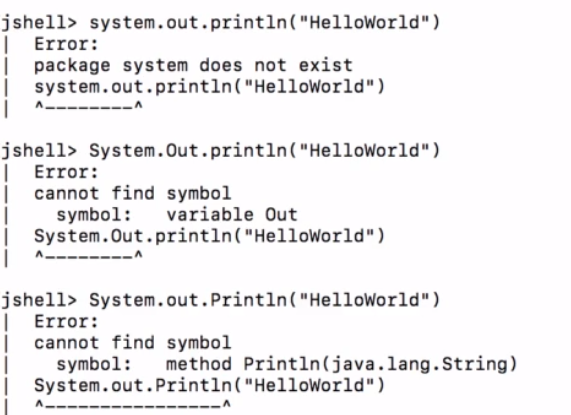
**Java**

**Java:** Object oriented programming language that is class-based and High level language.

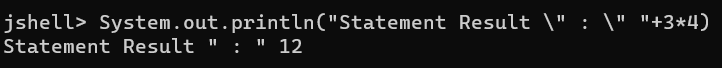
**Jshell:** REPL (Read Evaluate Print Loop) 🡪 Helps in faster evaluation of code written and was introduced in *Java 9* onwards.



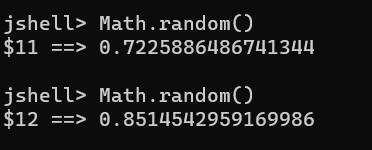
* 5\*5 🡺 Expression (5 🡪 Literal/Operands on which operations performed) (\* 🡪 Predefined functions performs some operation)
* **Precedence** 🡪 Higher priority 🡺 **()** ***\* / % > + - ++ --***
* *method\_path(parameter)* 🡪 System.out.println(3\*4) 🡪 *Statement*

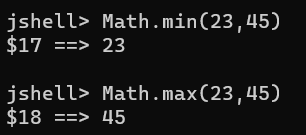


* Escape “ quotes with escape character \.*Escape characters : \n (new line) \t (tab) \ (Escape char)*

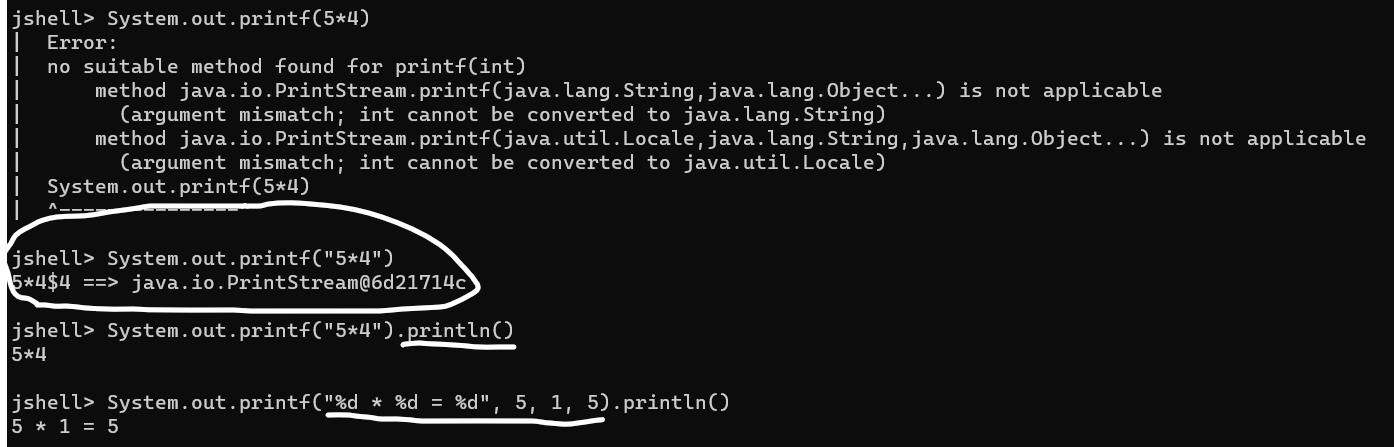


* ***Math.random()*** 🡪 Generates random value b/w 0 and 1. No parameter.
* ***Math.min()/Math.max()*** 🡪 Returns minimum/maximum of 2 variables. Require parameters.



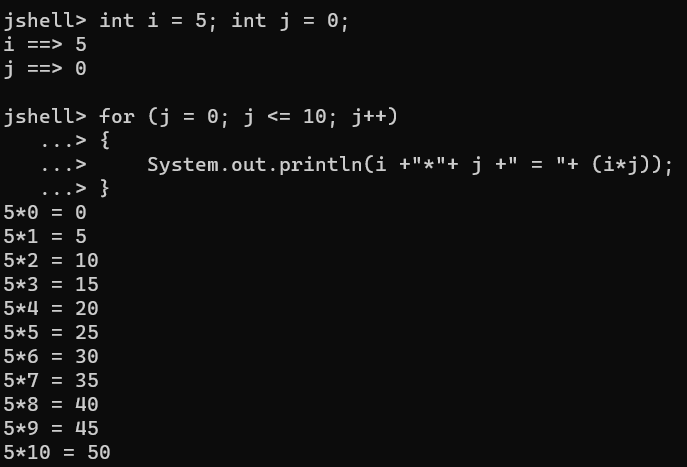
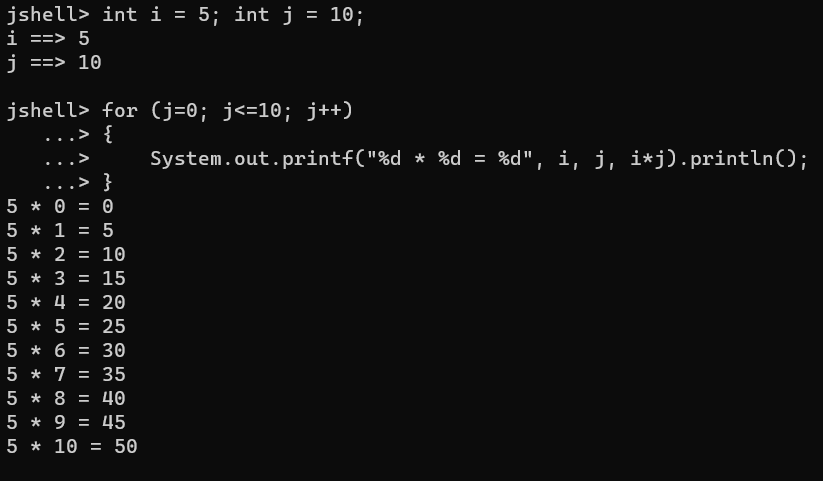


* ***System.out.printf()*** 🡪 It allows formatting to be printed which println() doesn’t support and also prints stream. For example : ***%d(Number), %s(String) and %f(Float)***.



* ***Challenge 1***:- *Multiplication table of 5 using Jshell*.

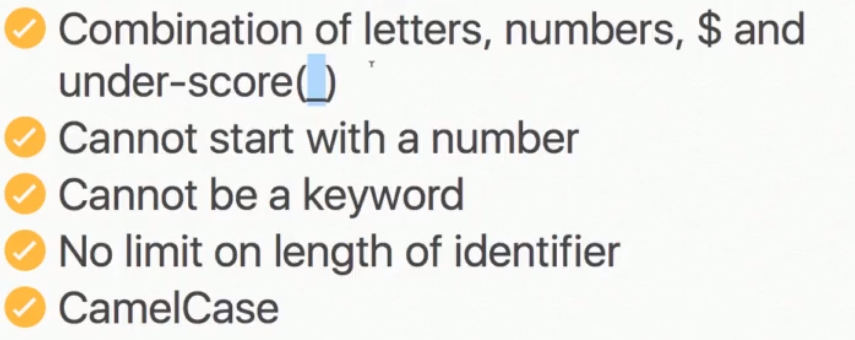
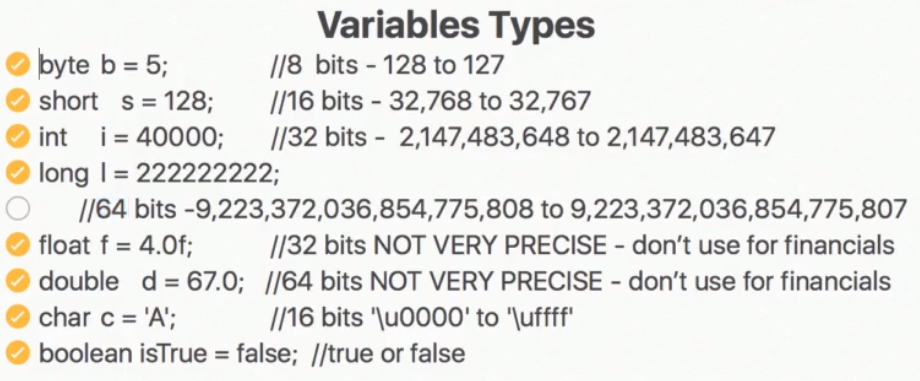
***Using System.out.println() Using System.out.printf()***

**Variables:** Variables are containers for storing data values. Variables have name, value and memory location. For example: int a = 100; int b = 110; int c = a + b;

***Primitive Data type vs Primitive Variable***: Basic data types build in Java and Variable that stores single value of Primitive data type.

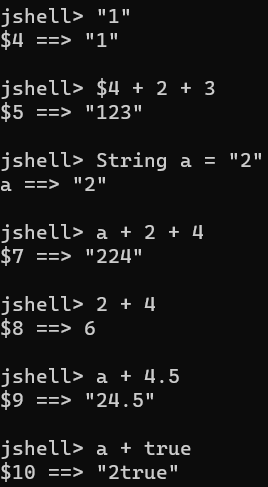
|  |  |  |
| --- | --- | --- |
| **Memory Location** | **Value** | **Name** |
| 5002 | 100 | a |
| 5006 | 110 | b |
| 5012 | 210 | c |

** **

Note: Float variable is by default double so we use ‘f’. Ex : float a = 4.5f; double b = 4.5;

**String Concatenation:-**

* If we add (+) one *String* with other *Short, Byte, Integer, Long, Float, Double, Boolean* result will be *String*.



**Operators:-**

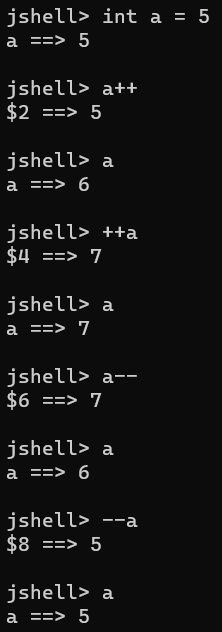
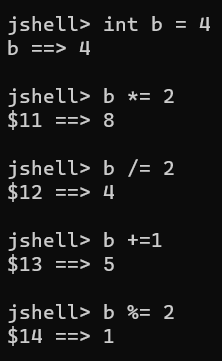
**= 🡪 Assignment Operator**

**==, <, >, <=, >= 🡪 Comparison Operator**

**&&,||, ^, ! 🡪 Logical Operator (Result is always true/false or boolean value)**

**-=, +=, /=, \*=, %= 🡪 Compound Operators**

**++, -- 🡪 Increment/Decrement Operator**

** **

**Conditions:-**

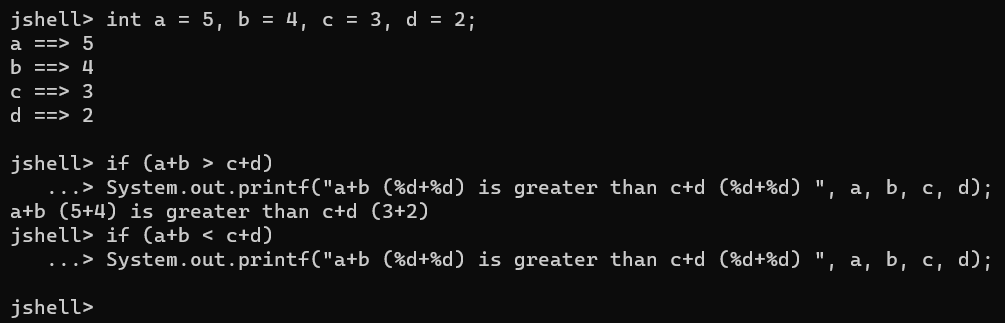
* **If Condition:** Executes ‘*If*’ inside statement only if condition is *‘true’*, otherwise no print.

*If(condition)*

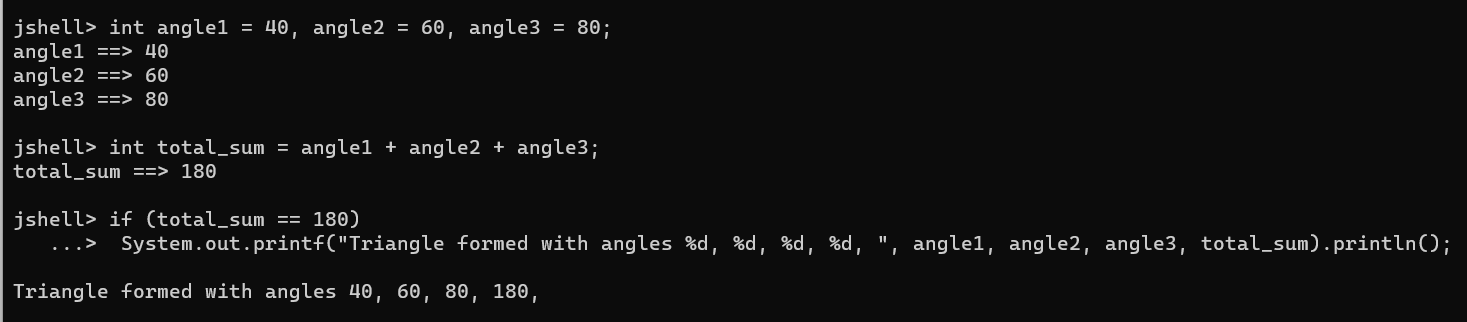
*{*

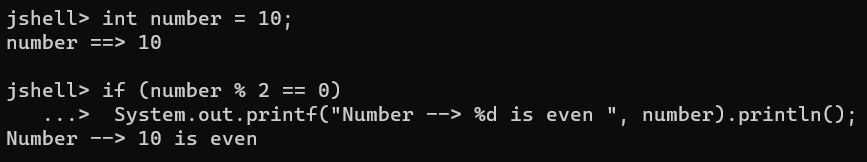
*Statement;*

*}*



* ***Challenge 2*:** *Sum of 3 angles if true print Triangle formed and check if number is even using Jshell.*

****

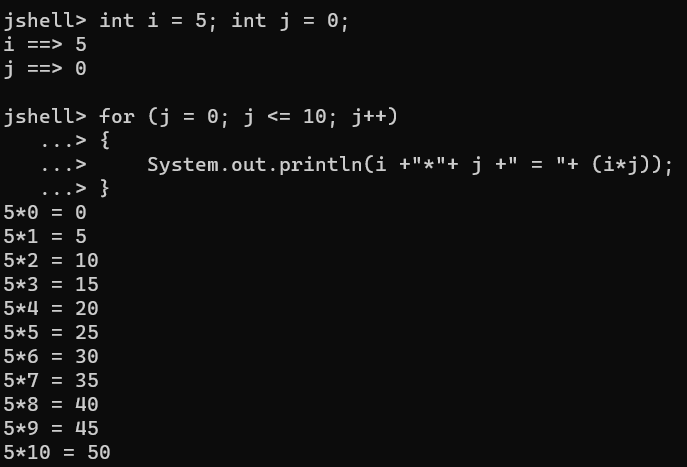
****

* **for loop:-** *for(initialization; condition; update)*

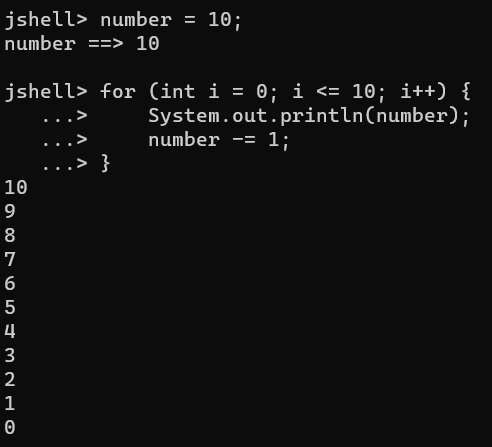
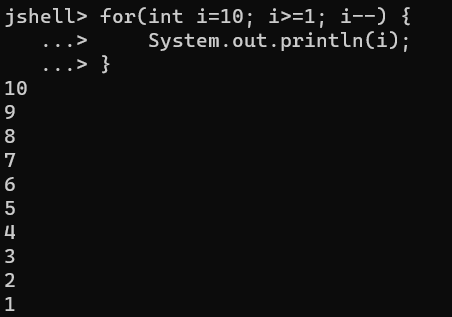
*{*

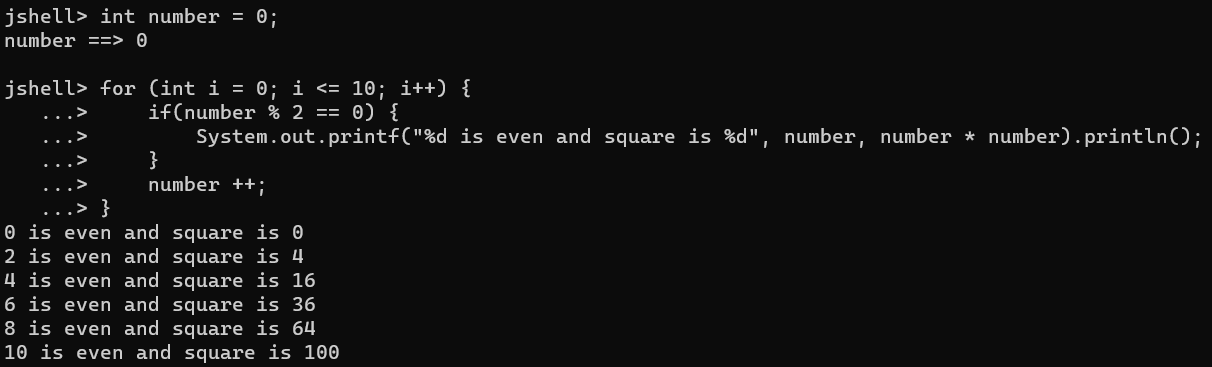
*Statement;*

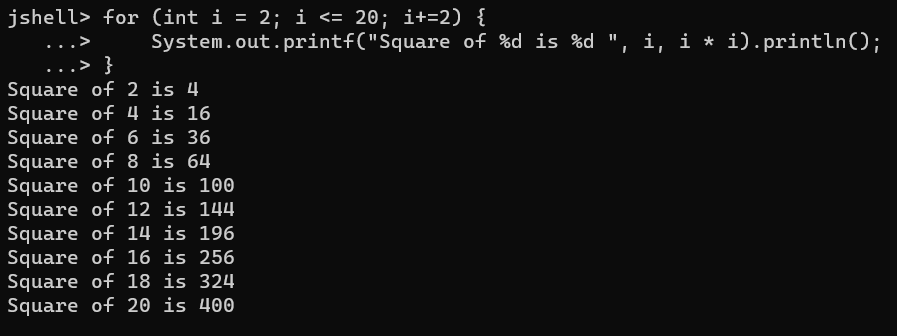
*}*

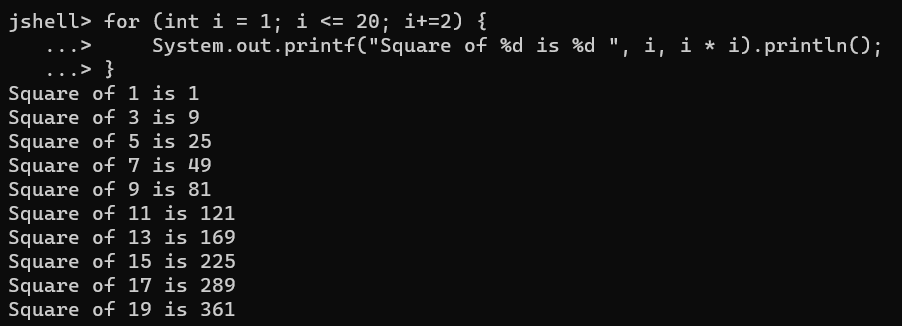


* **Challenge 3:** Print number in reverse from 10 to 0 and print first 10 number square if it’s even/odd using jshell.

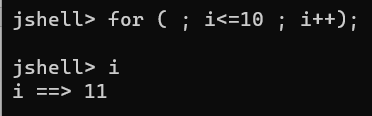
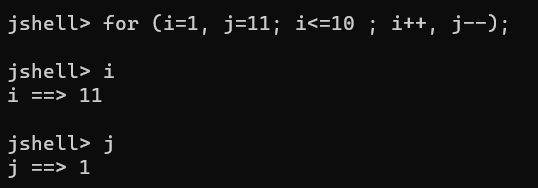
 



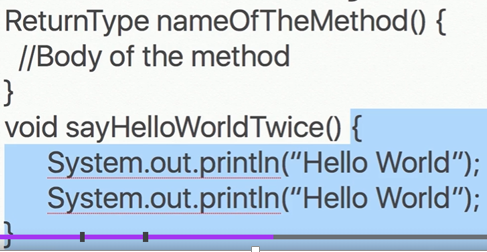
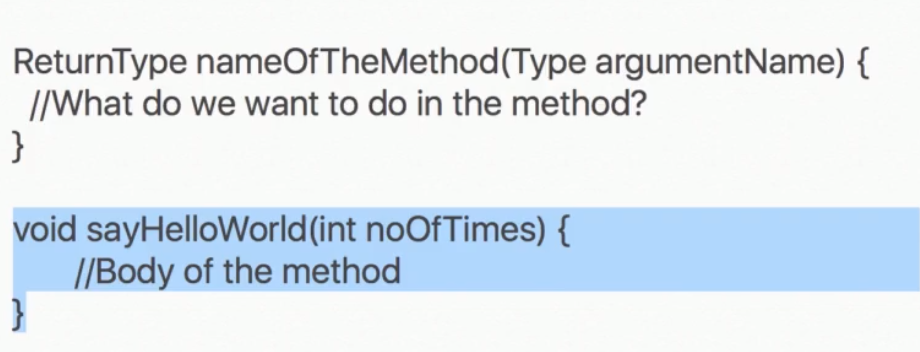




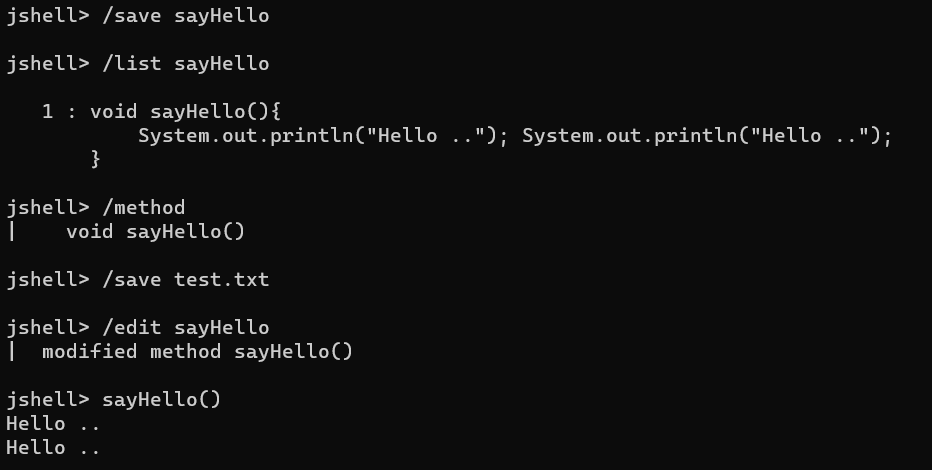
* ***Note:*** *for loop with “;” is an empty* *statement* that means even if we *don’t define* *initialization and update in for loop*, it can *execute statement* and can have *multiple variables but only one condition*. Example: *for( ; i <= 10; i++);* & *for(i=1,j=2; i<=10; i++,j--);* & *for(; ;); 🡪 Infinite loop.*

****** 

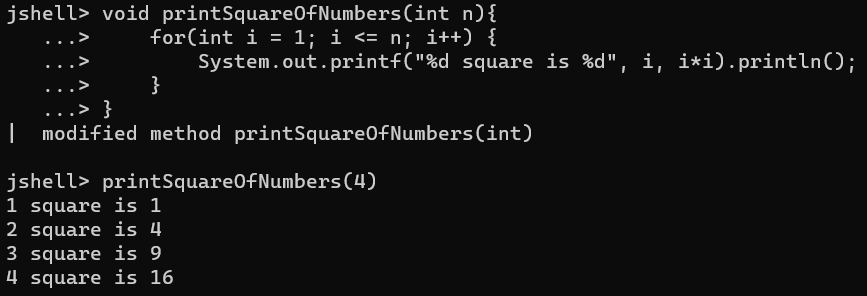
**Method:** It is a block of statements (code), which when called/invoked performs some action. It can be empty or have an argument or passing a parameter.

 ****

* **Jshell commands:** 
  + - * **/list 🡪** List the variables and methods. Ex: /list sayHello
      * **/save 🡪** Save the variables and methods.Ex: /save sayHello
      * **/edit 🡪** Edits variables and methods in Jshell editor. Ex: /edit sayHello

****

**Challenge 4:** Print square of numbers starting from 1 to n passed as an argument in a method.

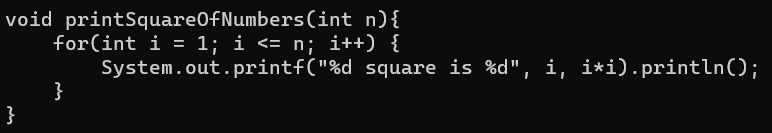


* ***Arguments vs Parameters****:* Arguments are the value passed when calling the method, whereas Parameters are defined inside a method. Ex:

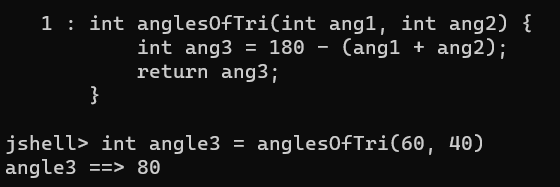
*Argument*

**

*Parameter*

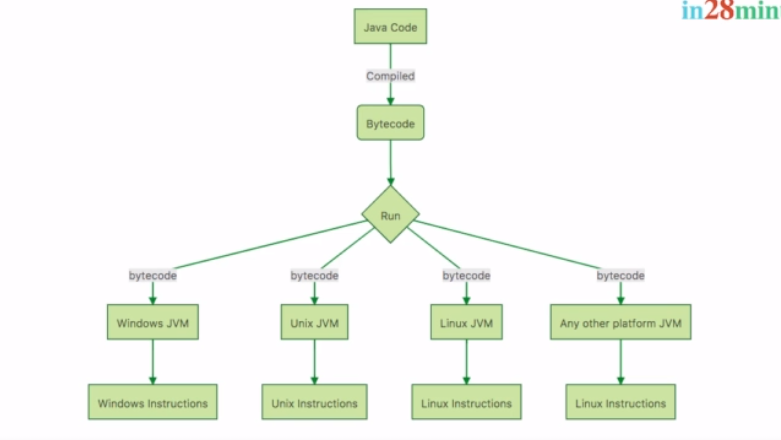
**

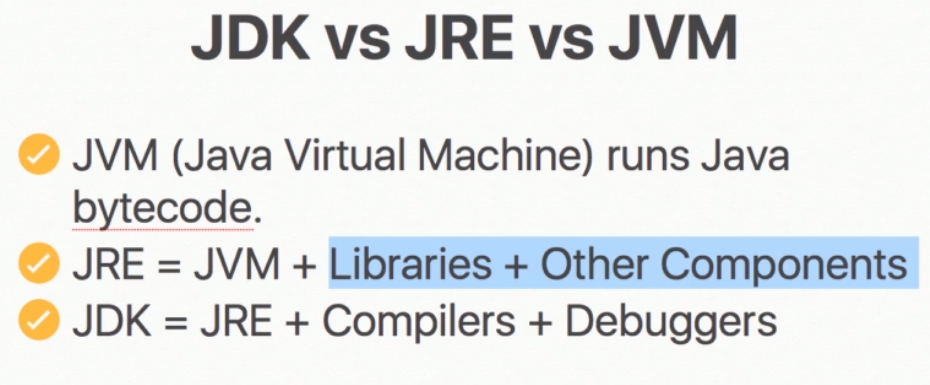
* *Method Overloading:* calling *abc()* and *abc(4)*, *abc(3,4)*. Same method having multiple parameters or arguments.
* *Method with returning value:* Must have a return type and data type defined for method and have only one return value.

**

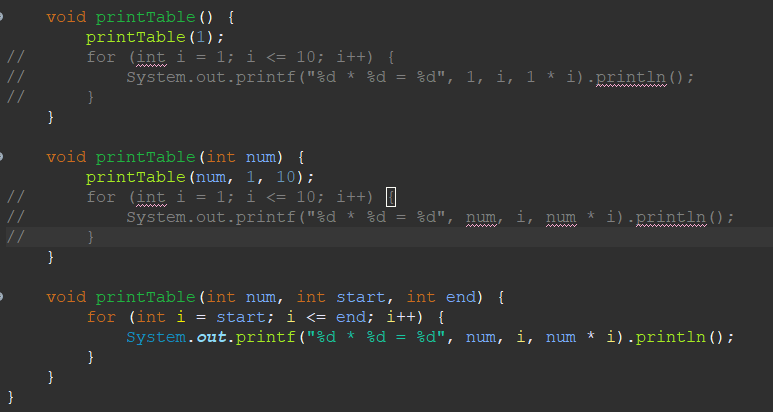
***Java, JRE, JDK***

* *Running a java code*
  + Java code/Java file which when compiled is converted to (.class file) byte code which ran by JVM to achieve platform independency apart from what OS we’re running in.
  + Windows/Linux/Unix JVM converts to OS instructions (0, 1 form) readable by machine.
  + *javac <filename>.java* 🡪 Converts java file .class
  + *java <filename>* 🡪 Runs the .class file

******

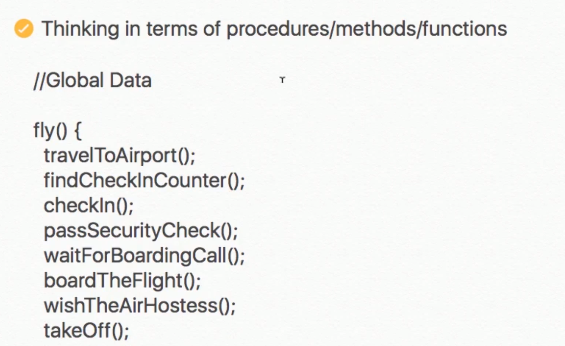
******

* In case if I am handing over a .class file created on Windows machine to Windows user, I only need JRE on other machine whereas if I am handing over .java file created on Windows machine to Unix user, Unix user will require a JDK, JVM is common in between both and is only used to compile Java file/code to bytecode or .class files.
* Jshell compiles your code automatically, but in case we need to write a code in java file then we need to have a *main() (public static void main(String[] args))* method and compile code every time we make changes in the file.
* *main() (public static void main(String[] args))* method can be empty but it will not do anything, if *no object created or method invoked* in main() method.
* *Code Refactoring :* Code refactoring is a practice to change your code structure in efficient way without changing it’s functionality.

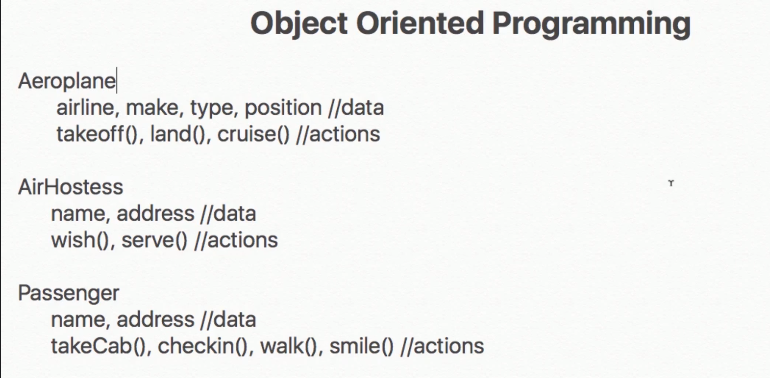


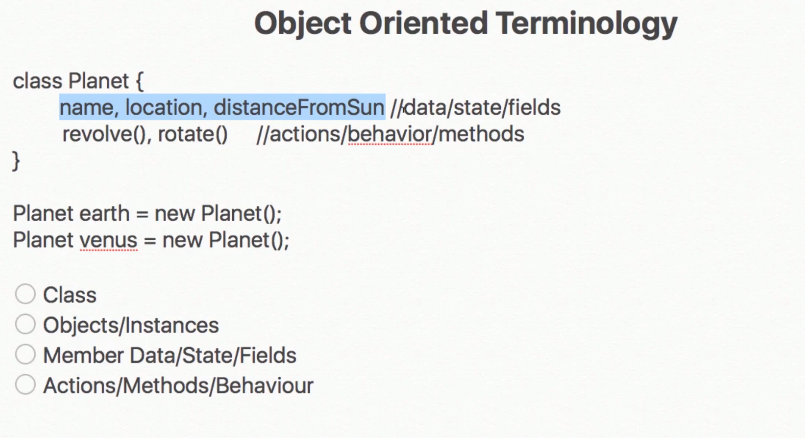
**Object Oriented Programming**

1. C/Pascal is a structured or procedural oriented programming, which is thinking of programming in terms of Structure, Procedure of method. In this we do not think what kind of data or state a method may have.



1. In Java structure of programming doesn’t matter, here we think about what kind of method it would be having and data/state of the method and actions it would be performing.

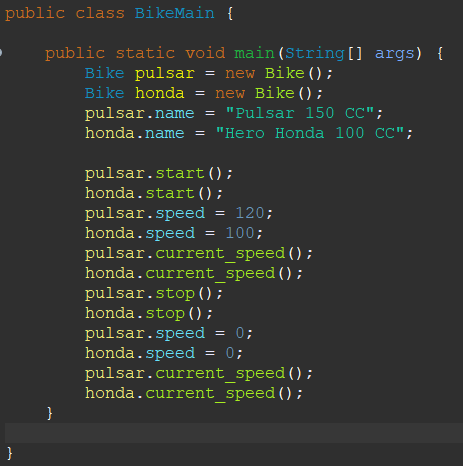


**

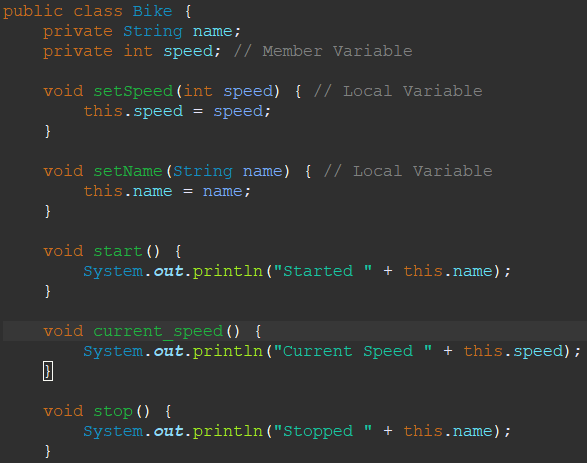
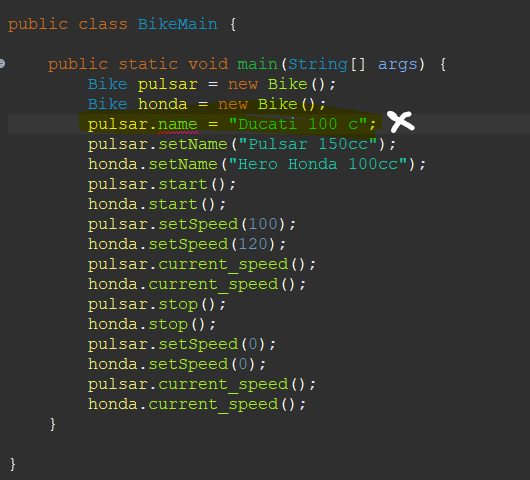
* *Class:* A class is template/blue print which defines like what kind of data it may contain and what will be the behavior/action of it (method) and share common properties and behaviours. For ex: For a planet, it needs to revolve and rotate.
* *Object:* An object is an instance of a class which can vary. For ex: distanceFromSun is different for Earth and Venus.
* *Member Data/State/Fields:* Can vary for each and every instance.
* *Actions/Methods/Behaviour:* Action it needs to perform on it’s object.

*Syntax:*

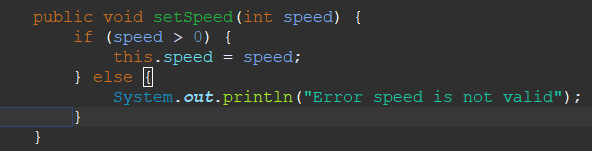
* *Class- class <ClassName>{}*
* *Object- <ClassName> nameOfObject = new <ClassName>();*
* *Calling method- <nameOfObject>.method\_name(arguments);*
* *Calling variable or setting value- <nameOfObject>.variable = value;*

**

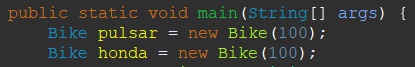
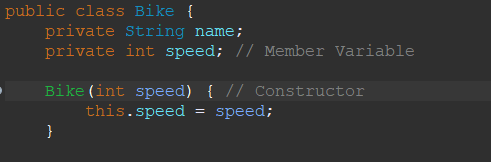
* *private:* Once we make a variable private in a class, it’s value can’t be updated from outer class, which is good practice. The values can be updated using getter/setters or methods and *this* keyword is used to set value of local variable to Member variable.

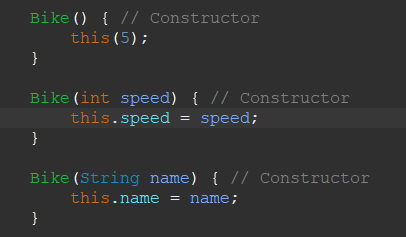
* *

* ***Encapsulation****: Encapsulation refers to the bundling of fields and methods inside a single class. It prevents outer classes from accessing and changing fields and methods of a class.*
  + We can also put validations for *invalid data changes*. For ex: if we set the speed of bike to -100 that should be invalid, what we can do is go into Bike class and put checks for speed > 0.

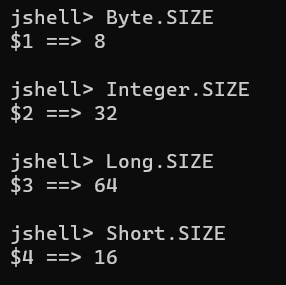
* *

* ***Abstraction****: Abstraction is a process of hiding the implementation details and showing only simplified view to the user.*
* ***Abstraction vs Encapsulation****: Abstraction and encapsulation go hand in hand. Encapsulation is a process of bundling everything in a single entity, whereas abstraction is hiding the complexity of a program.*
* ***Constructor****:* It is method which is used to initialize a variable using objects. It is method having no return type and is same as class name. By default java provides a default constructor but in case if you created a constructor yourself, then the default constructor won’t work and it will throw an error, to resolve this you need to create a empty constructor yourself and invoke other constructors if you want using *this* to initialize value. Important note is *this* can call only one constructor.

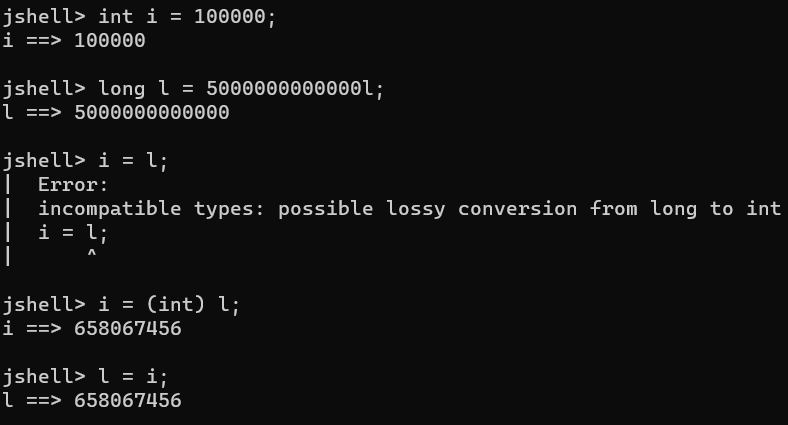
* *

* *

* *Wrapper classes in Java: The wrapper class in Java is used to convert primitive data types into objects. MAX\_VALUE, MIN\_VALUE, SIZE.*

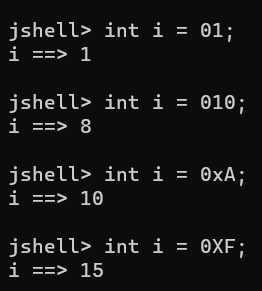
**

* *Casting: We can only convert lower data type to high data type using explicit conversion. Otherwise Java will automatically convert lower data type to higher data type which is called implicit conversion. Ex :*

**

* *Java supports binary, octal, hexadecimal representation. Anything starting with 0, 0x or 0X.* 
  + *Binary – 0 and 1*
  + *Octal – 0,1,2,3,4,5,6 and 7*
  + *Hexadecimal – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F*

*For ex:*

**

* *Postincrement/Postdecrement: Assignment operator first then increment/decrement.*
* *Preincrement/Predecrement: First increment/decrement then assignment operator.*

*jshell> int j = --i;*

*j ==> 9*

*jshell> i*

*i ==> 9*

*jshell> int i = 10;*

*i ==> 10*

*jshell> int j = i--;*

*j ==> 10*

*jshell> i*

*i ==> 9*

*jshell> int i = 10;*

*i ==> 10*

*jshell> int j = i++;*

*j ==> 10*

*jshell> i*

*i ==> 11*

*jshell> int i = 10;*

*i ==> 10*

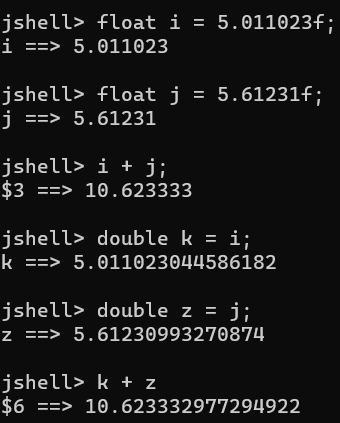
*jshell> int j = ++i;*

*j ==> 11*

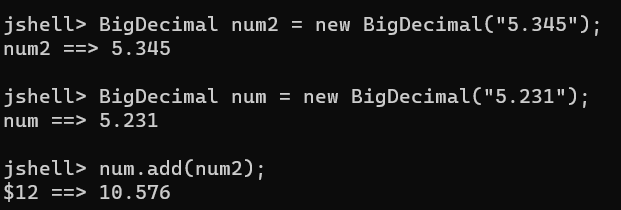
*jshell> i*

*i ==> 11*

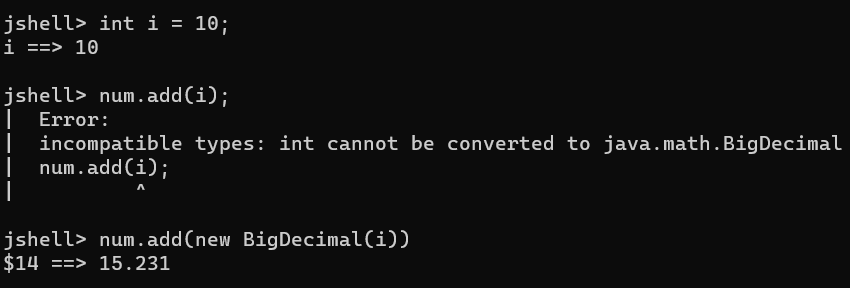
* *BigDecimal:* When we perform arithmetic operations on float and double, result come are inaccurate. It’s good for representation but not for operations, so we use BigDecimal. BigDecimal is a class in java and provide operations as method.

**

* ***Syntax:*** *BigDecimal <objname> = new BigDecimal(“<decimalnumberasString>”);* If we don’t use String number it will again be not accurate.

**

* BigDecimal goes with BigDecimal, otherwise you need to do type conversion using BigDecimal class constructor or import java.math.BigDecimal class.

**

* *Relational Operator :* Result is always Boolean value, true or false.

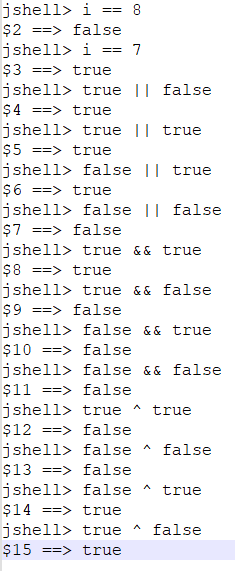
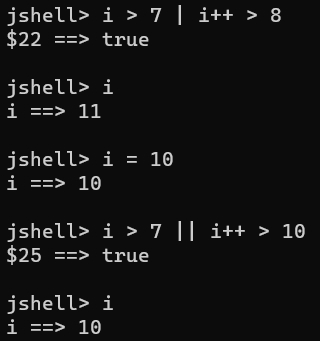
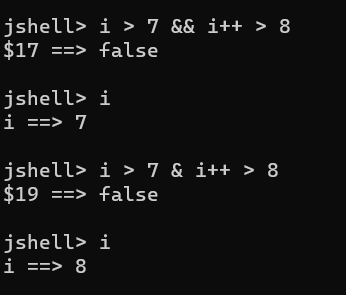
*&& (AND) 🡪 Both condition needs to be true otherwise will return false. Won’t check both conditions. Notice increment operator won’t changes value first and then compares.*

*|| (OR) 🡪 Either one of condition should be true otherwise will return false. Won’t check both conditions. Notice increment operator won’t changes value first and then compares.*

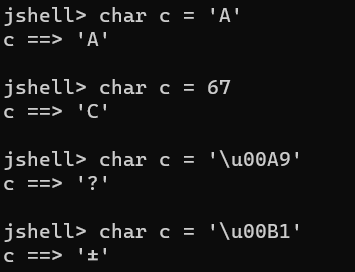
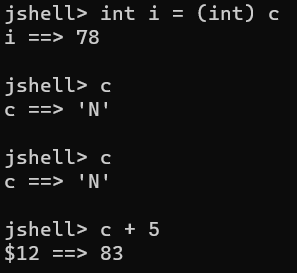
*^ (XOR) 🡪 Both conditions if different then return true otherwise will return false.*

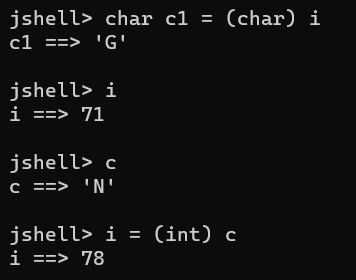
*& 🡪 Executes for both condition and then evaluate if true, then true otherwise false. Notice increment operator changes value first and then compares.*

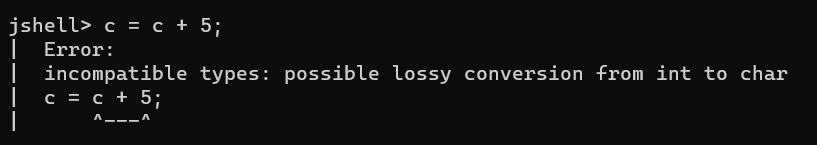
*| 🡪 Executes for both condition and then evaluate if either one true then true otherwise false. Notice increment operator changes value first and then compares.*

*  *

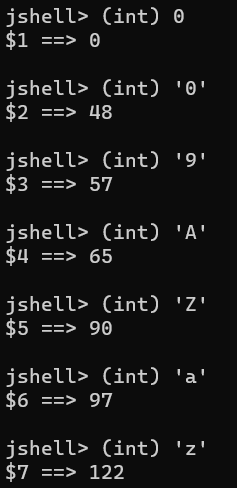
* *Character:* Used to represent a single character. Character in Java supports*, ASCII (65) and Unicode (‘/u0000’)* and we can perform integer operations and typecasting. Remember char is *1 byte* and int is *4 bytes*.
* *toUpperCase(): Character.toUpperCase(ch)*
* *toLowerCase(): Character.toLowerCase(ch)*

* *

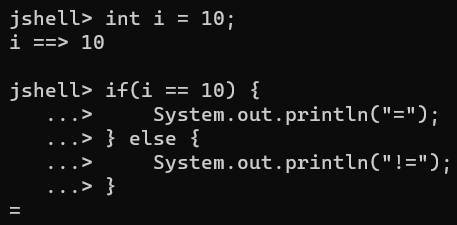
**

**

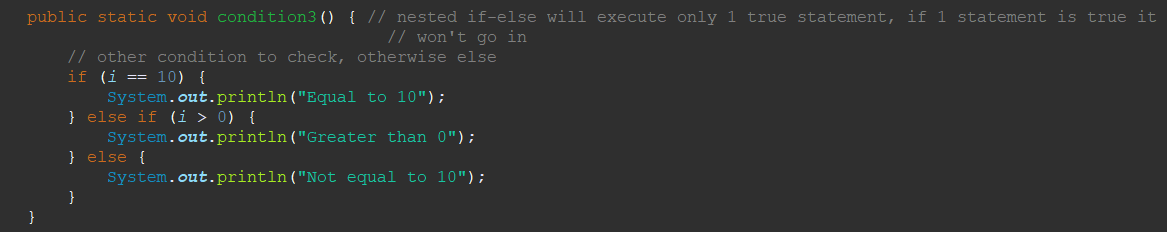
* *To check if a char is digit between 0 and 9, then we need to have AND condition with ASCII values of 0 and 9 char.*

**

* *Conditional programming:*
  + *If-else: On true execute if statement, otherwise else statement.*

**

* + *If-else if-else: Only one true condition statement is executed in nested if-else, otherwise else statement.*

**

* *Scanner Class: Scanner class is in java.util.Scanner, used for taking user input.*

*Syntax: Scanner <objname> = new Scanner(System.in); //System.in is taking user input*

*int I = <objname>.nextInt(); //takes int value*

* *Switch case: If we don’t put break;, it will execute statements after true onwards. Switch is only applicable for int(short, byte, int), String(char), Enum.*

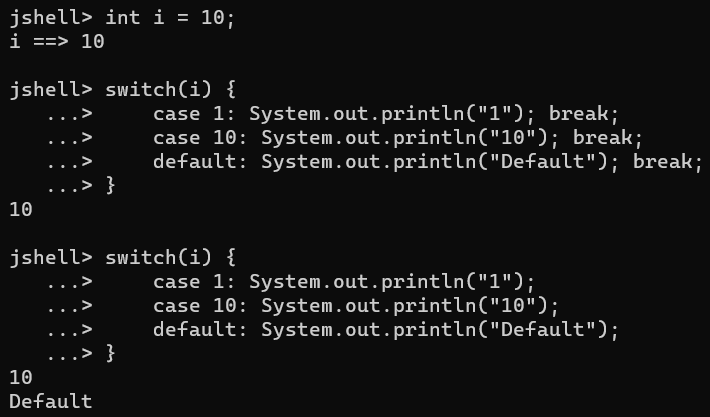
*Syntax: switch(case) {*

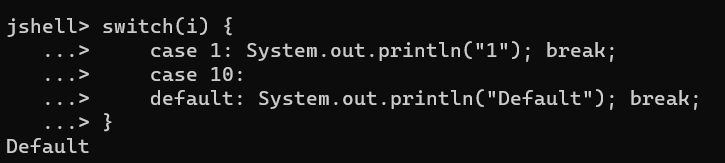
*Case1: <stmt>; break;*

*Case2: <stmt>; break;*

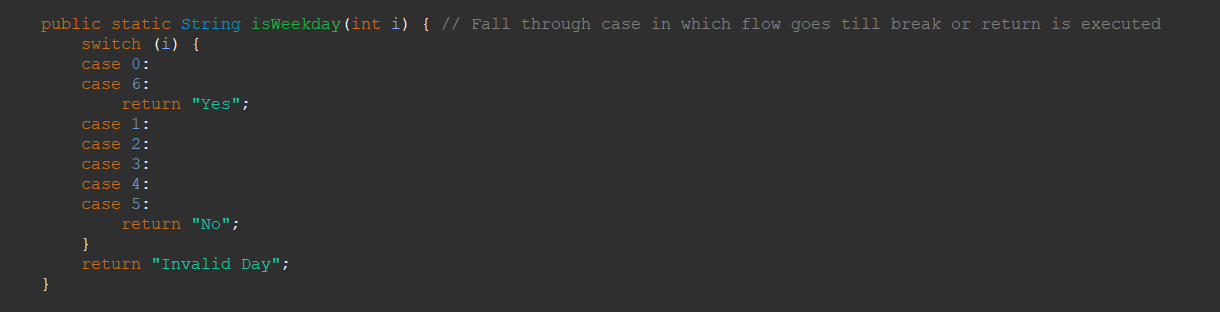
*default: <stmt>;break;*

*}*

**

**

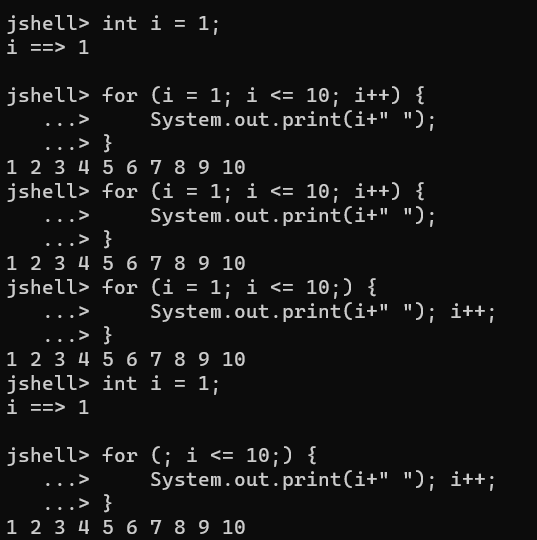
* *Fall through case in switch statement is if we didn’t defined case statement, then it will execute till next break and return statement. In case if we put return, defining break is not mandatory.*

**

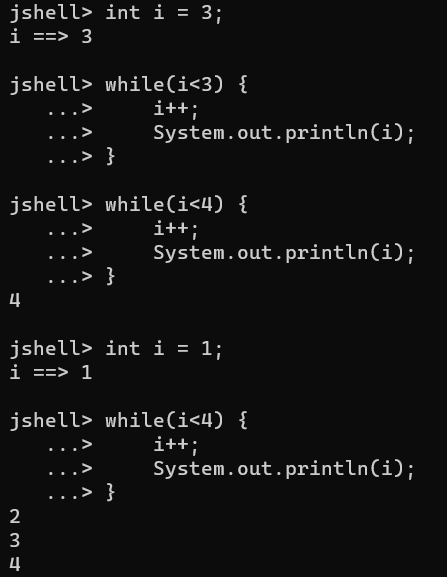
***Loops***

* *Loops are used to run same code again and again based on condition.*
* ***For Loop****: It is not mandatory to have initialization, condition or update to be provided in for loop. It can still execute the code.*

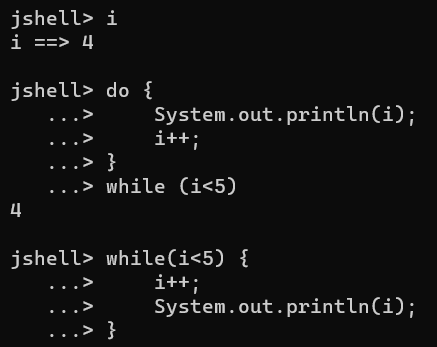
*Syntax: for(initialization; condition; update) {<Stmt>}*

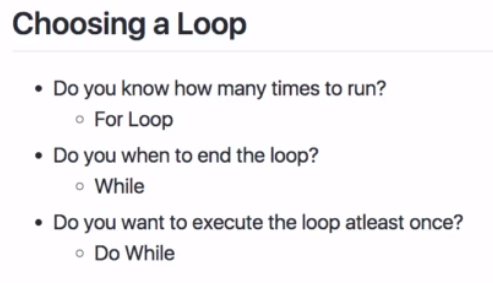
**

* ***While Loop:*** *While will execute statement till it’s condition is true, once false it exits.*

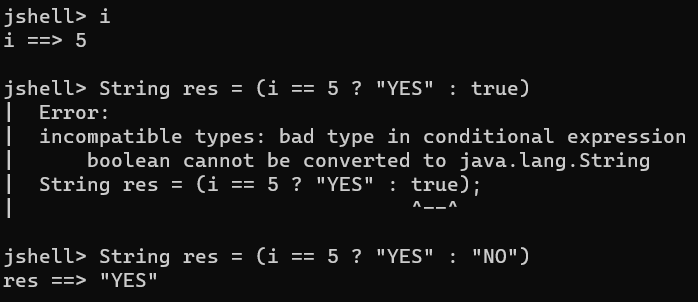
******

* ***doWhile Loop:*** *Condition check is at last in do-while loop. Statement in do block will at least be executed once since condition is checked at last.*

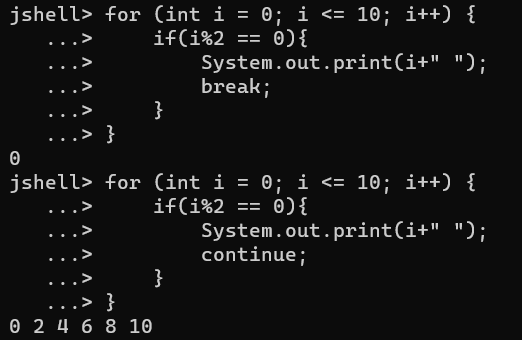
******

******

* ***Ternary Operator:*** *condition? <stmtfortrue>: <stmtforfalse>. <stmt> should be of same data type.*

******

* ***Break vs Continue:*** *Break will exit out of loop once condition is false. Continue will continue executing next statement but will skip the condition where it fails.*

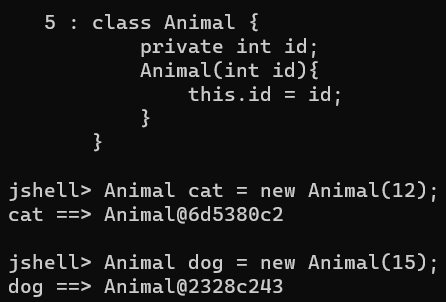
******

* ***Reference Type and Reference Variables:*** *Any class is reference type and it’s objects are reference variables, variables created are primitive data type.*

*Each method is stored in a Stack memory and objects are stored in Heap memory. For primitive data type values can be compared on basis of values, whereas for reference variable to be equal, they need to have same memory reference.*

|  |  |
| --- | --- |
| Location | Object |
| *1A* | *Animal12* |
| *1B* |  |
| *1C* | *Animal15* |
| *1D* |  |

|  |  |  |
| --- | --- | --- |
| Location | Value | Var |
| *A* | *5* | *i* |
| *B* | *1A* | *dog* |
| *C* | *1C* | *cat* |
| *D* | *1C* | *cow* |

**

Heap

Stack

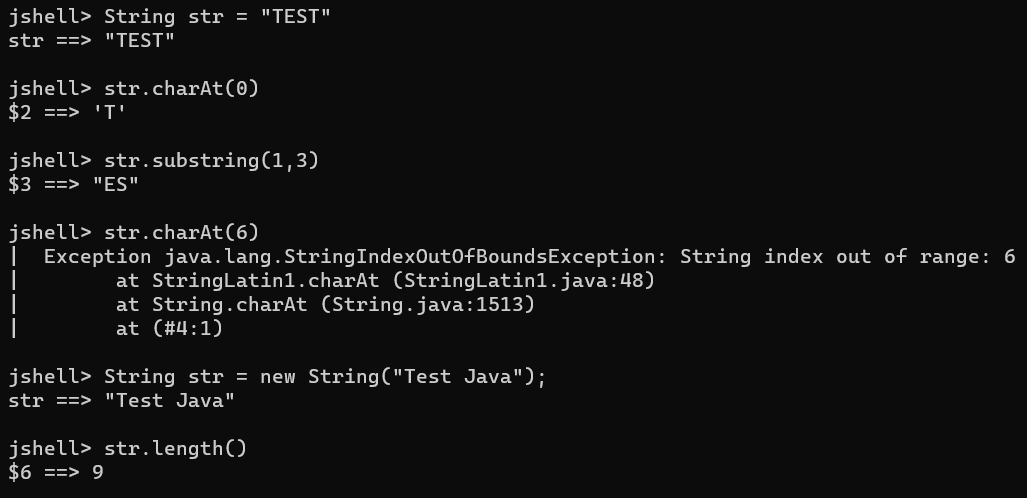
* *cow = cat (Reference location is copied into value and are equal, Since Object variable they’re referring to same object)*
* ***String:*** *String is a special class in java and an object to store character sequences.*

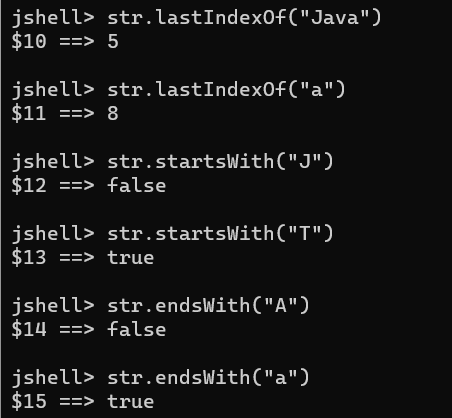
*Ex: String str = “<Value>”*

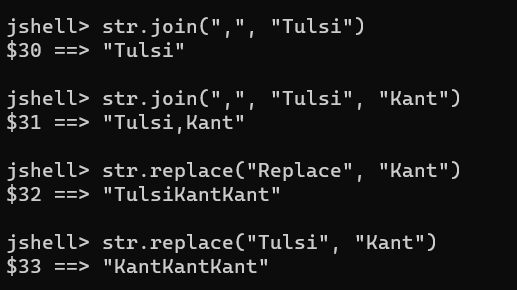
*String str = new String(“<Value>”);*

***Operations on String:***

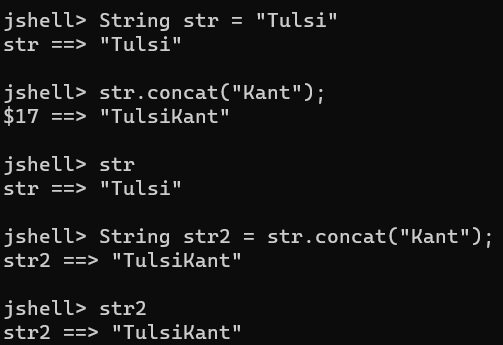
1. *Str.length() 🡪 Gives length of String*
2. *Str.charAt(i) 🡪 Gives char at the index value*
3. *Str.substring(I,j) 🡪 Gives characters b/w start index and last index.*
4. *Str.equals(“str”) 🡪 true/false matching case string if equal.*
5. *Str.equalsIgnoreCase(“str”) 🡪 true/false w/o matching case if equal.*
6. *Str.lastIndexOf(“str”) 🡪 returns last occurrence index of char in string.*
7. *Str.indexOf(“str”) 🡪 returns first occurrence index of char in string.*
8. *Str.startsWith(“str”) 🡪 true/false if string is starting with matching string.*
9. *Str.endsWith(“str”) 🡪 true/false if string is ending with matching string.*
10. *Str.concat(“str”) 🡪 Adds string at end.*
11. *Str.contains(“str”) 🡪 If string contains the string.*
12. *Str.toLowerCase(“str”) 🡪 Converts String to lower case.*
13. *Str.toUpperCase(“str”) 🡪 Converts String to upper case.*
14. *Str.join(i,j,k) 🡪 String will join first char with other char separated via ‘,’.*
15. *Str.replace(“st1”,“str”) 🡪 Replaces string with next string value.*

******

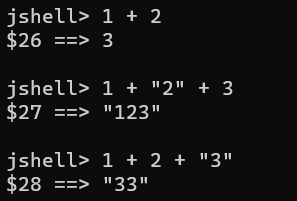
*** ***

******

* ***Immutable String:*** *String is immutable in java, which means it’s value can’t be updated in same String object. The string previous values will remain same as it is in heap memory, since String is an object but a new String object will be created with values pointing to old String reference.*

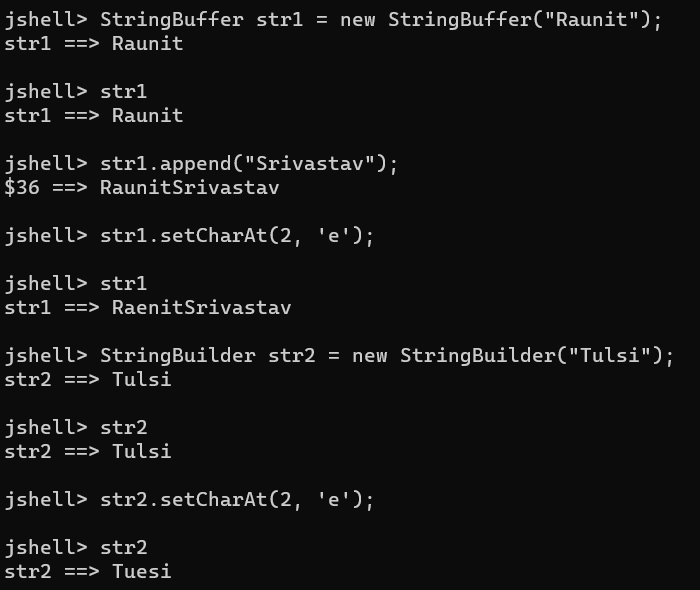
******

* ***If string is + with integer the result will be string as + will behave as concatenation then.***

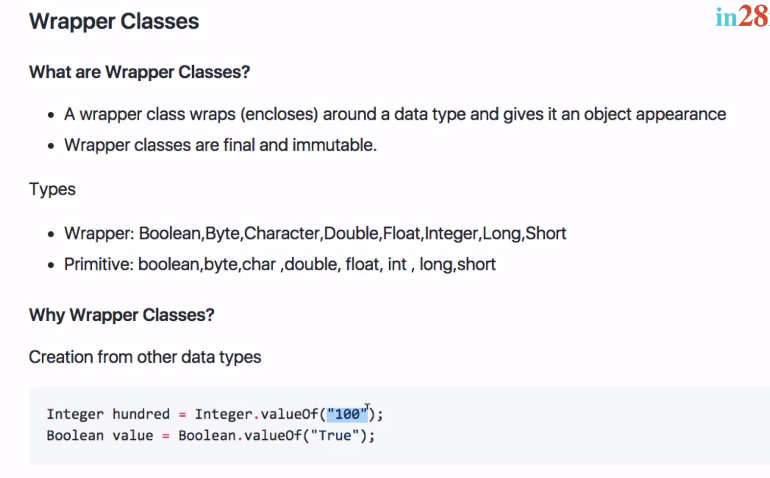
******

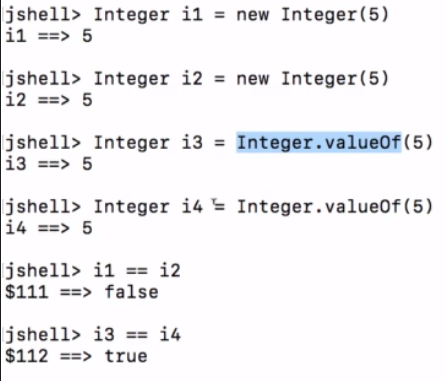
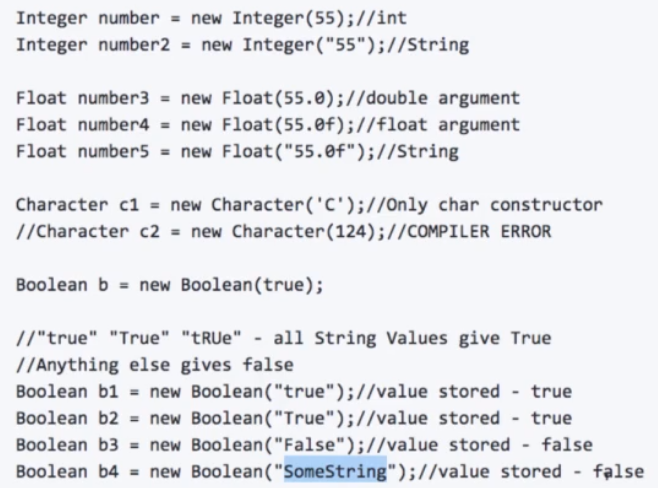
* ***String Buffer/String Builder:*** *String Buffer and String Builder is mutable. String buffer is not thread safe and not recommend for multiple strings whereas String builder is thread safe.*
* ***Syntax:*** StringBuffer sb = new StringBuffer(“Str”);

StringBuilder sb = new StringBuilder(“Str”);

****

* ***Wrapper Classes:*** Wrapper classes offer more options than primitive classes. Ex: *valueOf(),MAX\_VALUE,MIN\_VALUE,BYTES,SIZE. Deprecated now.*



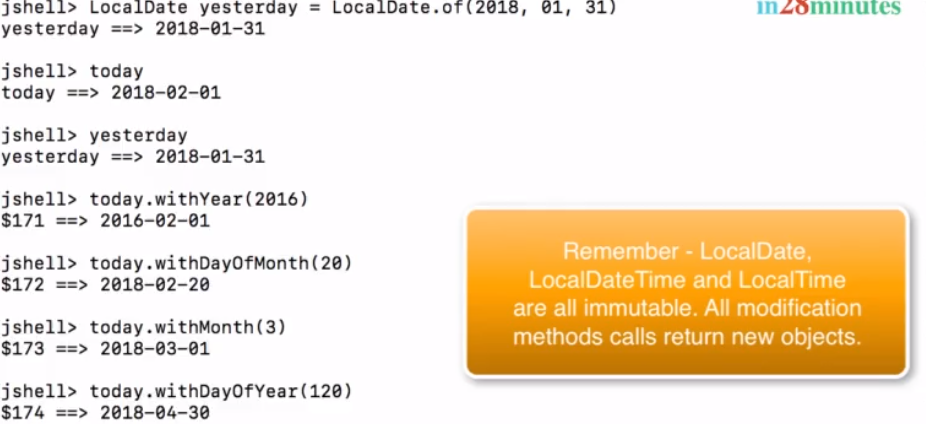
 

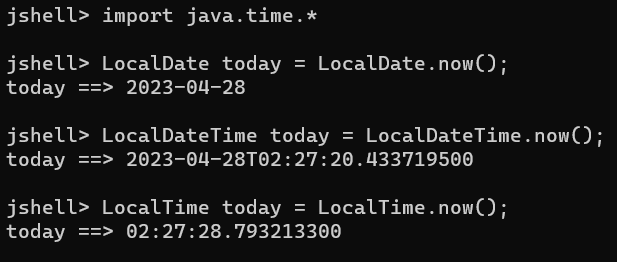
* ***LocalDate, LocalDateTime, LocalTime:*** *Returns current date, time, date and timestamp.*

Syntax: *LocalDate today = LocalDate.now();*

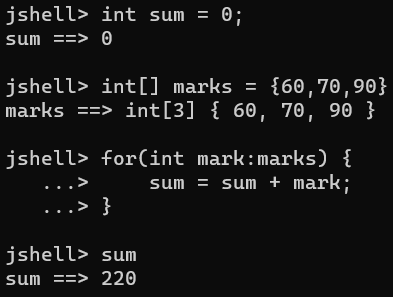
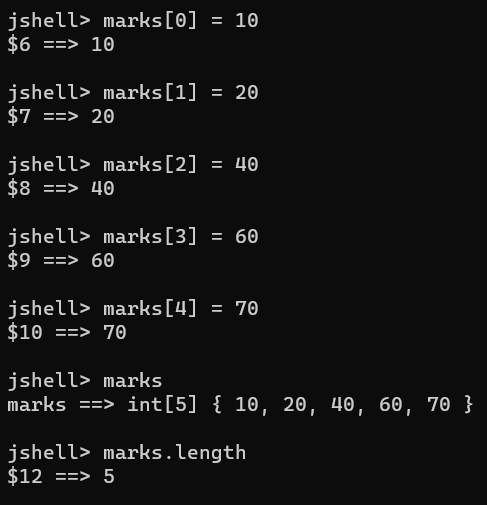
*LocalDateTime today = LocalDateTime.now()*

*LocalTime time = LocalTime.now()*

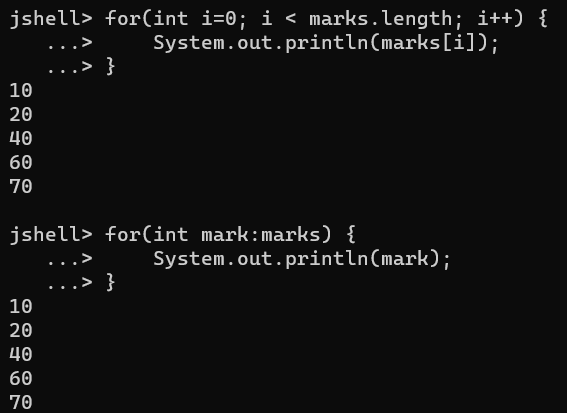
******

******

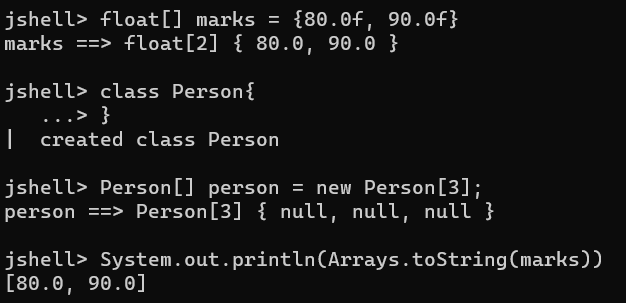
* ***Arrays:*** *Arrays are used to store same data type elements. Array can contain 0 element or N elements in it. Length of array is 0 to N-1.*

****** ******

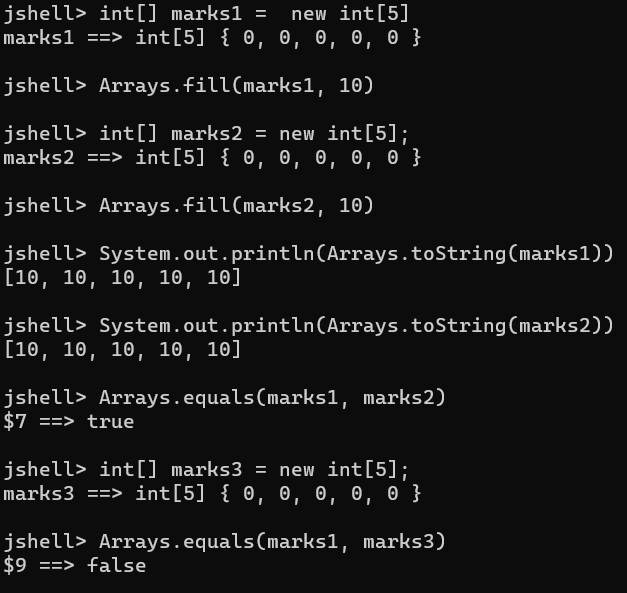
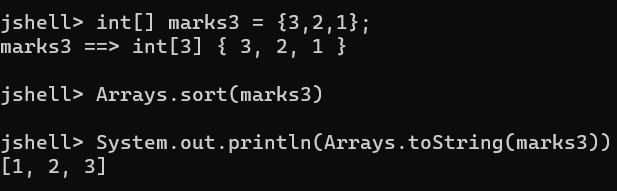
* ***Accessing/Reading Array elements:*** Loop for the array elements. If we need to print array on console, we can use *System.out.println(Arrays.toString(<arr>));* as well.

******

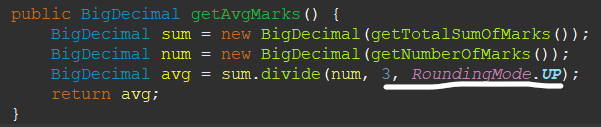
*Array storing class*

******

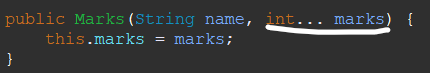
* ***Arrays Operation:*** 
  + *Arrays.toString(<Arr\_Name>):* Use to print array in console.
  + *Arrays.fill(<Arr\_Name>,<Value>):* Fills all array elements with same value.
  + *Arrays.equals(<Arr\_Name1>,<Arr\_Name2>):* Compares two arrays if equal or not.
  + *Arrays.sort(<Arr\_Name>):* Sort all arrays elements in ascending order.

* *

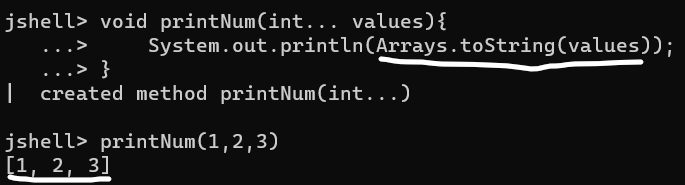
* ***BigDeciamal RoundingMode.UP/DOWN:*** BigDecimal if values is like 9.66666676676 🡪 It can be rounded up like *(new BigDeciamal(num, pos, Mode)).* 
  + ***UP 🡪*** Means the decimal will be rounded to upper value. Ex: 9.666667 🡪 9.66.
  + ***DOWN 🡪*** Means the decimal will be rounded to lower value. Ex: 9.666667 🡪 9.67.

******

* ***Variable Arguments:*** In case we don’t have exact max length of parameters to be defined, we can user Variable arguments. Variable arguments should be the last parameter defined in method. Variable arguments are treated as arrays. Ex: int… marks

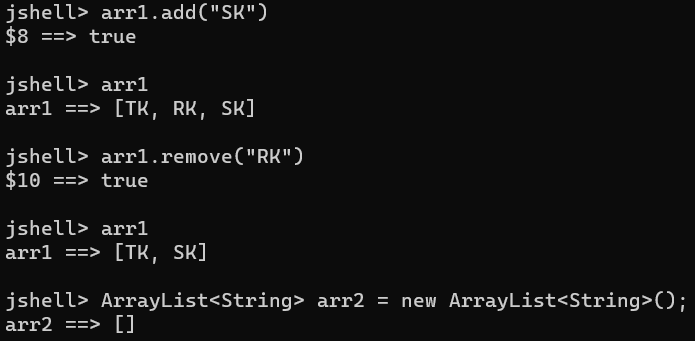
******

******

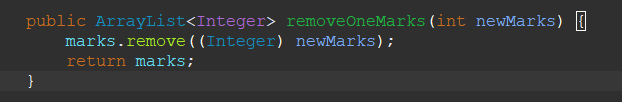
******

* ***ArrayList:*** Arrays have fixed size, whereas ArrayList has dynamic size. Adding and removing in array is complex process and only way is to copy elements to new array and add/remove element in new array. So the concept of arraylist comes into picture.

***Syntax:*** *ArrayList<DataType> name = new ArrayList<DataType>();*

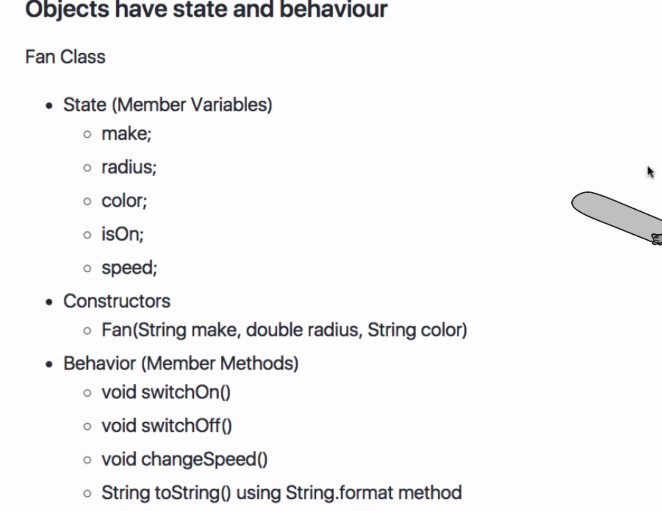
**

* ***Note:*** *If arrayList is Integer type then it will remove at index and not element. To remove element we need to typecast it as.*

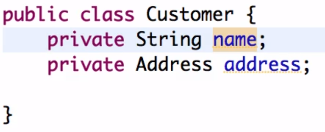
******

***OOPS***

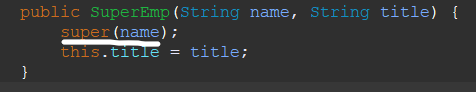
Objects have state (values) which changes accordingly, variables are member variables of a class and methods changes the behavior of an object.



* ***Object Composition:*** When an object contains reference of another object, it’s called object composition or in simple words. A class having object of another class within it and an object created for the class.

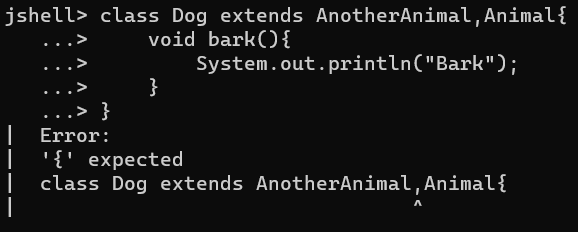
******

* ***Inheritance:*** Inheriting the property of another class is called Inheritance, inheritance reduces code duplication and increase code reusability. *Subclass extends Superclass* and when subclass object is called it gets all properties of Superclass variables and methods.
* ***Object class is at top of inheritance:*** Every class in java *extends Object class. Java by default extends class Object for methods like toString(), hasCode() e.t.c. If we use our toString() method we’re overriding the method of default Object class.*
* ***Method Overriding:*** Method overriding is a process in which a subclass can override or implement method of a Superclass. *First superclass constructor gets called then subclass constructor.*
* *If we create a constructor in super class and in subclass extending it. It will throw error Implicit super constructor SuperPerson() is undefined. Must explicitly invoke another constructor. So we need use super keyword along with parameter we need to pass to superclass. Java by default creates a constructor with super().*

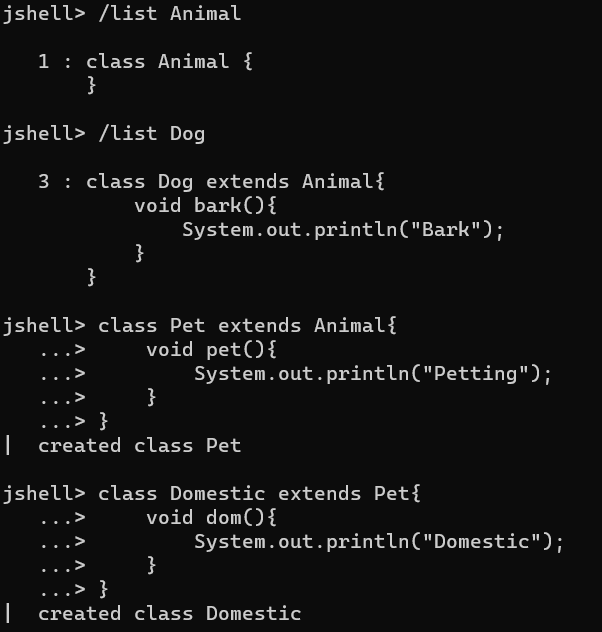
******

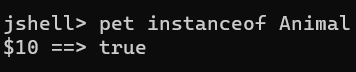
* ***Java doesn’t allow multiple inheritance as it complexes the process of inheritance. But we can have superclass, subclass, innersubclass… and a superclass having multiple subclass, which is called inheritance chaining. We can use instanceof to know if subclass is an object of a superclass or not.***

*Multiple Inheritance*

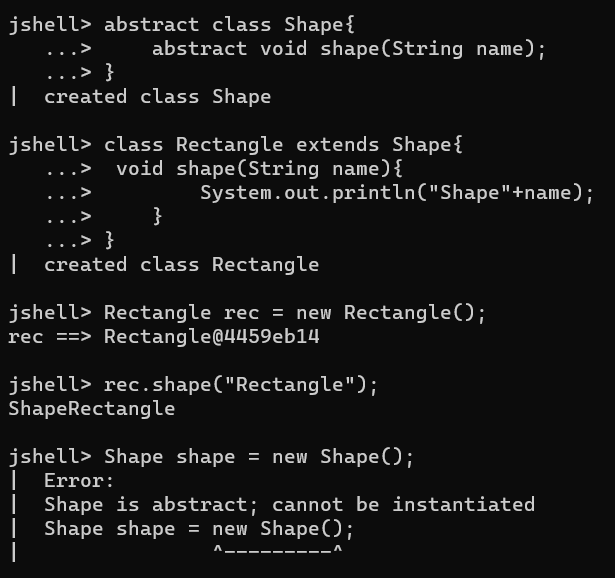
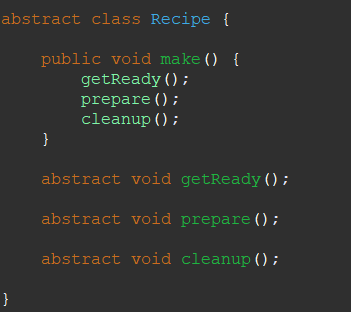
******

*Inheritance Hierarchy*



******

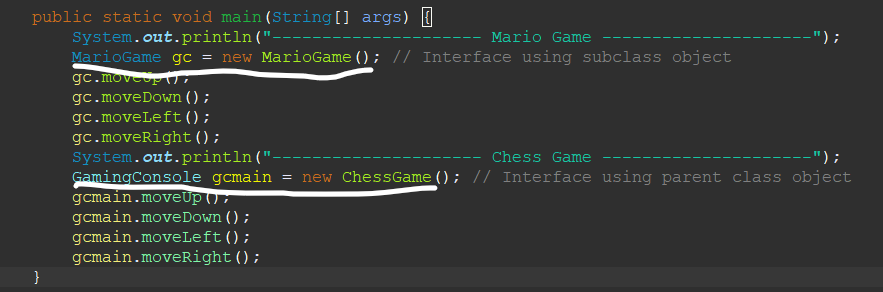
* ***Abstract Class:*** *Abstract class only contains implementation and not body of any method, the empty body of abstract class mandatory to be implemented by it’s subclass when extended. We can’t create object of an abstract class. Abstract class will have abstract methods only. Abstract can have no methods as well. Abstract class can have variables and methods with implementation but since object can’t be created for abstract class, subclass object can call the method of abstract class.*

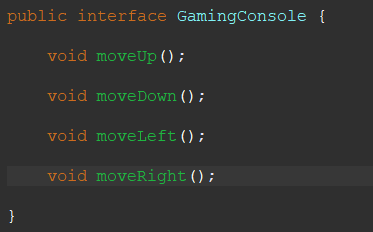
****** ******

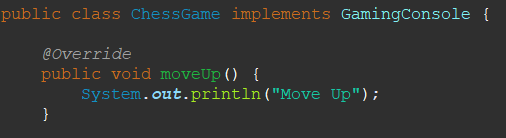
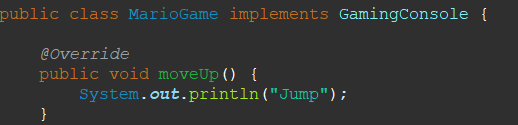
* ***Interface:*** *Interface in many terms is similar as that of abstract class, it is also used to define unimplemented methods, which then once implemented by subclasses methods.*

*In interface we can use Superclass object along with subclass objects as well. Interface keyword is used to represent interface class and implements keyword by subclass to implement superclass methods.*

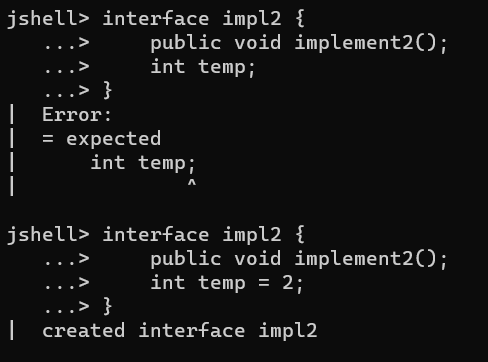
*Interfaces in practical scenario can be used if any dummy implementation we have we can use for now, but in future if someone designs real implementation, then all we need to do is change in object to new class object, which makes easy to implement.*

******

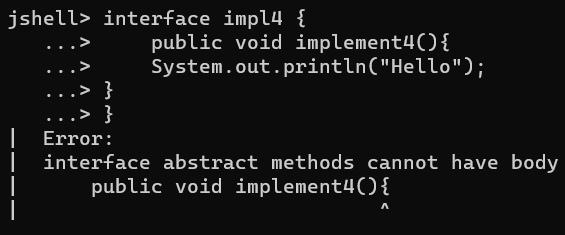
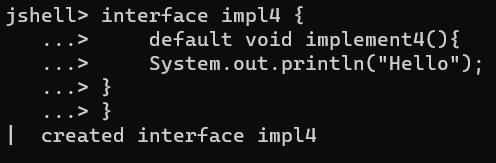
******

*** ***

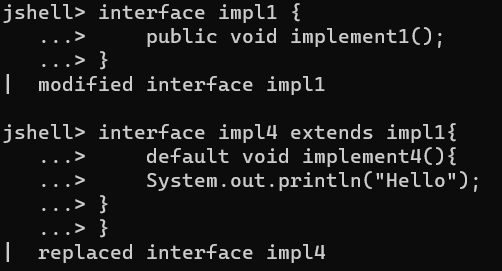
* ***Can interface have variables?*** *No, it needs to be defined value.*

******

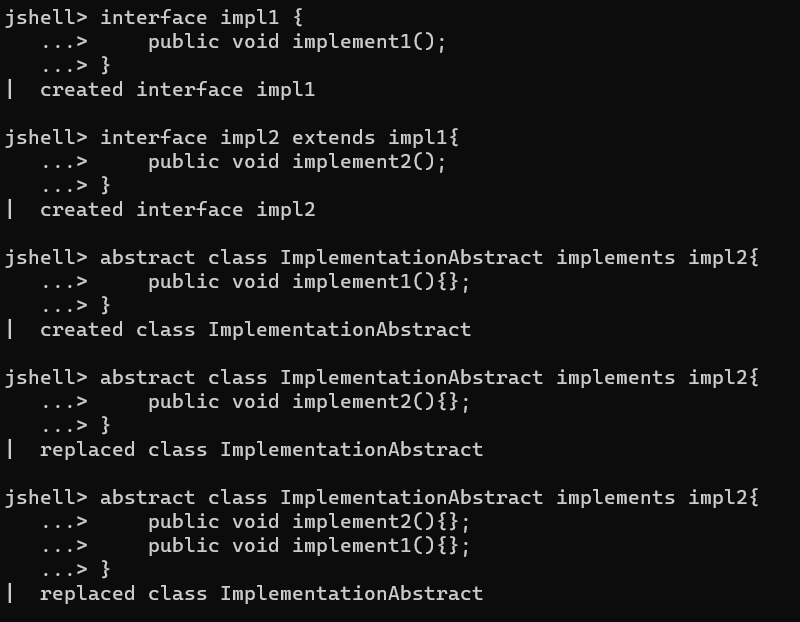
* ***Can interface have normal method?*** *No normal methods are not allowed,**We can use default keyword before method to have default implementation in interface.*

****** ******

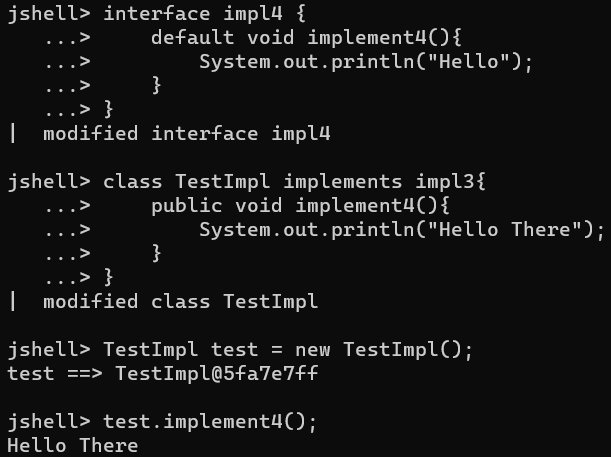
* ***Can interface extends each other?*** *Yes, but if a 3rd class is implementing extending interface, it needs to implement both interface methods.*

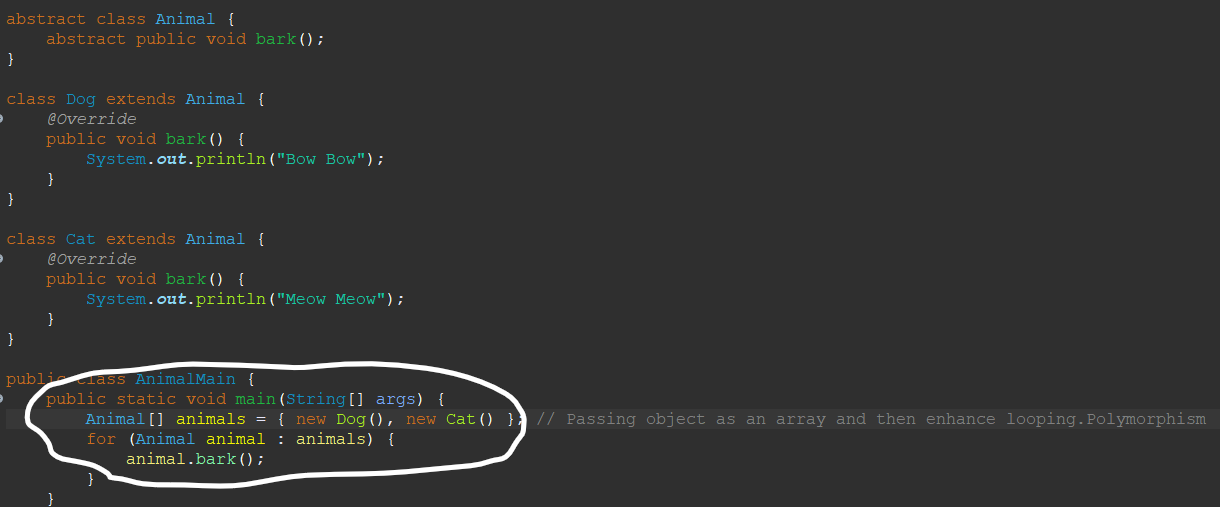
****** ******

* ***What if when extending interface I need to implement only one interface method and not of other interface?*** *We can use abstract method in that case.*

******

* ***Can we override default method of an interface when implemented?*** *Yes we can.*

******

* ***Abstract class vs Interface:*** 
  + *Abstract class can have variables with different values, but in interface values are constant.*
  + *Abstract class can have all types of variable/methods whereas in interface variable/methods should be public it can’t be private.*
  + *All Abstract class methods need to be implemented by it’s subclass, but in interface the class implementing it does not need to implement all methods.*
  + *Multiple Abstract class cannot be extended, whereas multiple interfaces can be implemented.*
* ***Polymorphism:*** *Same code with different behavior based upon input. Interface and Abstract classes by default uses concept of polymorphism. Ex: For Dog class bark method prints “Bow Bow” but for Cat class same method prints “Meow Meow”.* ****