JAVA BACKEND DEVELOPMENT PROGRAM

Java SE Basic

OUTLINE

- Variable and Primitive Data Type
- Comment
- Operator
- Flow control
- Keywords

VARIABLE

- A variable is a named memory location capable of storing data

- ***Object variables*** refer to objects, which are created by instantiating classes with the new operator

- We can also store data in ***simple variables***, which represent data only, without any associated methods

VARIABLE

- Declaration
    - Data Type
    - Name
        - int noOfWatts;
- Initialization
    - Name
    - Value
    - noOfWatts = 100;
- Combined:
    - int noOfWatts = 100;

VARIABLE

- Data types
- Primitive data types: built in types
- Non primitive data types

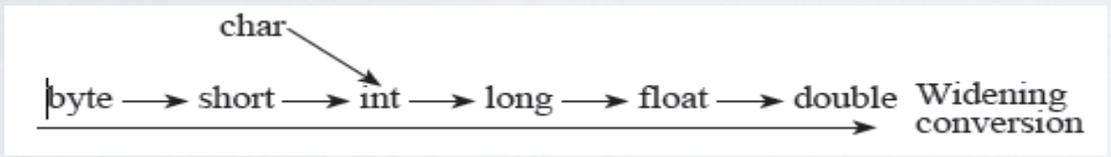
VARIABLE

- Primitive Data Types

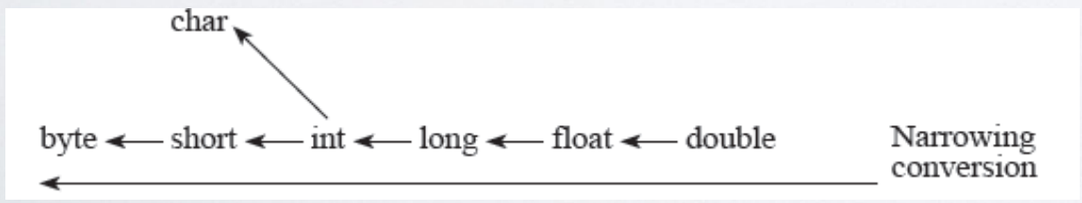
Data Type	Default Value	Size	Range
byte	0	8	–128 to 127 (inclusive)
short	0	16	–32,768 to 32,767 (inclusive)
int	0	32	–2,147,483,648 to 2,147,483,647 (inclusive)
long	0L	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive)
float	0.0F	32	1.401298464324817e–45f to 3.402823476638528860e+38f
double	0.0D	64	4.94065645841246544e–324 to 1.79769313486231570e+308
char	'\u0000'	16	0 to 65535
boolean	false	Not defined	true or false

CONVERSION AND CASTING

- **Conversion,** also known as widening conversion, are performed automatically
  - A smaller box can be placed in a bigger box and so on.
  -



- **Casting** also known as narrowing conversion.
  - A bigger box has to be placed in a small box.
  - Casting is not implicit in nature.
  - int i = (int)(8.0/3.0);
  - Casting will lose precision
  -



CONVERSION AND CASTING

```java
int i = 5/2;

int m = (int)(5.0/2.0);
double n = (int)5.0/2.0;

int k = (int)2147483648.0f; //Integer.MAX_VALUE=2147483647

char c = (char)75; //ASCII: 75->K
```

WRAPPER CLASS

- A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

- Integer, Character, Short, Long, Double

STRING

- String is a sequence of characters. In java, objects of String are **immutable** which means a constant and cannot be changed once created.

- String Pool — a collection of Strings which are stored in heap memory

STACK MEMORY
HEAP SPACE

- Stack : Static memory allocation and the execution of a thread

- Contains primitive values

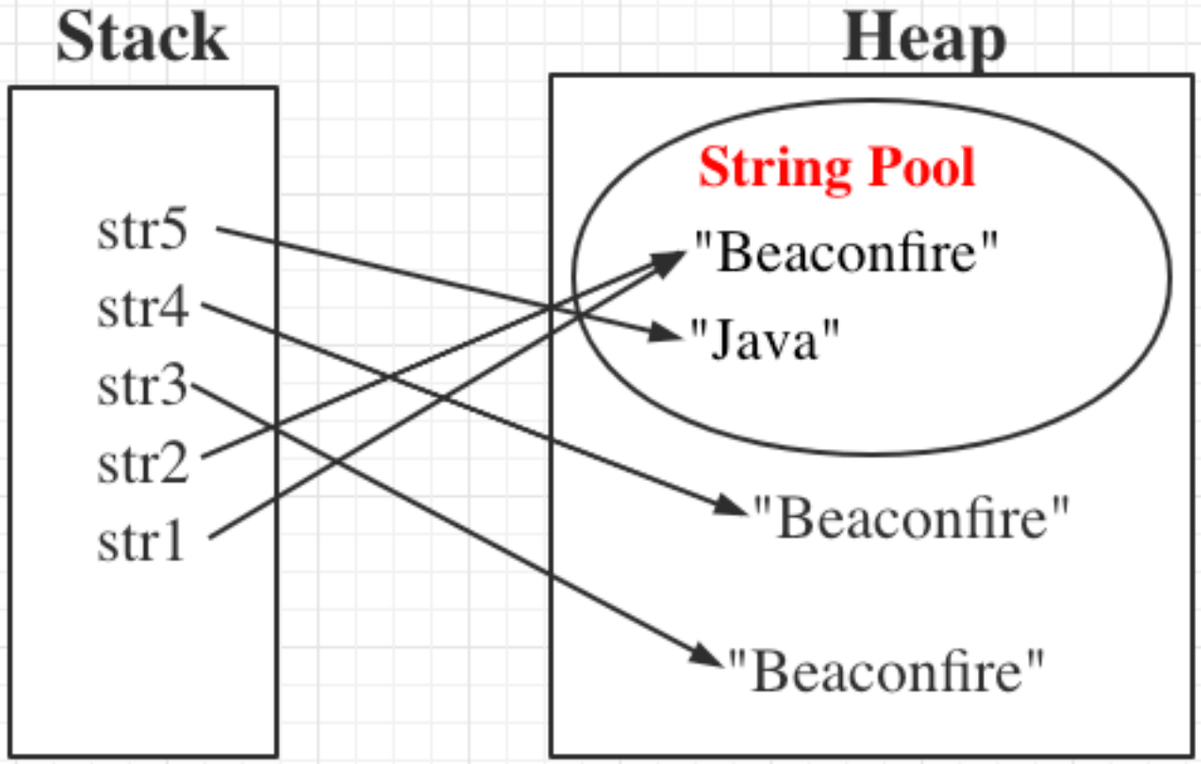
- References to objects that are in the heap

STACK MEMORY
HEAP SPACE

- Heap : dynamic memory allocation for Objects
    - Objects can be globally accessed.

STACK MEMORY
HEAP SPACE

```
String str1 = "Beaconfire";
String str2 = "Beaconfire";
String str3 = new String("Beaconfire");
String str4 = new String("Beaconfire");
String str5 = "Java";

str1==str2; //true
str1==str3;//false
str3==str4;//false
```



- String Pool: Save memory, reusability(don't need to create a new String if already exists)

COMMENT

- Java supports three types of comments
  - 1. /* text */
    - The compiler ignores everything from /* to */.
  - 2. //text
    - The compiler ignores everything from // to the end of the line.
  - 3. /** text */
    - This is a documentation comment and in general its called doc comment. The JDK javadoc tool uses doc comments when preparing automatically generated documentation.

OPERATOR

- Arithmetic operations in Java
  - Precedence: (*, /, %) > (+, -)
  - Parentheses: evaluate the innermost parenthesized expression first, and work your way out through the levels of nesting
  - No {} or [ ] in parentheses in Java

OPERATOR

```
int x = 1, y = 2, z;

z = x + y * 2;      //5

z = (x + y) * 2;    //6
```

OPERATOR

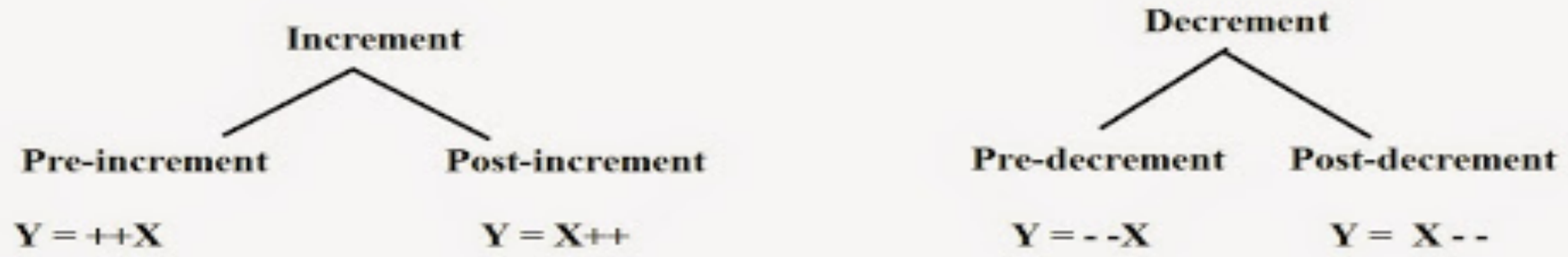
```
int x = 5, y = 2, z;

z = x / y;      //2

z = x % y;    //1
```

COMPOUND ARITHMETIC/ ASSIGNMENT OPERATORS

Operator	Use	Meaning
`+=`	`x += 1;`	`x = x + 1;`
`-=`	`x -= 1;`	`x = x - 1;`
`*=`	`x *= 5;`	`x = x * 5;`
`/=`	`x /= 2;`	`x = x / 2;`
`%=`	`x %= 10;`	`x = x % 10;`

INCREMENT AND DECREMENT OPERATORS

Increment and Decrement Operators



Expression	Intial Value of X	Final Value of X	Final Value of Y
Y = ++X	4	5	5
Y = X++	4	5	4
Y = - -X	4	3	3
Y = X - -	4	3	4

LOGICAL OPERATOR

Operator	Name	Type	Description
!	Not	Unary	Returns true if the operand to the right evaluates to false. Returns false if the operand to the right is true.
&	And	Binary	Binary operation: logical and 0001 & 0011 = 0001 = 1
|	Or	Binary	Binary operation: logical or 0001 | 0011 = 0011 = 3
^	Xor	Binary	Binary Operation: exclusive or 0001 ^ 0011 = 0010 = 2
&&	Conditional And	Binary	If the operand on the left returns false, it returns false without evaluating the operand on the right.
||	Conditional Or	Binary	If the operand on the left returns true, it returns true without evaluating the operand on the right.

RELATIONAL OPERATORS

Operator	**Result**
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

FLOW CONTROL

- Selection Statements
  - If and switch
- Iteration Statements
  - While, do-while, for and nested loops
- Jump Statements
  - Break, continue and return

SWITCH

```
char c = 'A';
switch (c) {
case 'A': {
    System.out.println("A");
}
case 'B':{
    System.out.println("B");
}
default:
    System.out.println("Default");
}
```

A
B
Default

```
switch (c) {
case 'A': {
    System.out.println("A");
    break;
}
case 'B':{
    System.out.println("B");
    break;
}
default:
    System.out.println("Default");
}
```

A

IF

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

TERNARY EXPRESSION

- Variable = (condition) ? expressionTrue : expressionFalse

```
boolean valid = true;
int i;
if(valid) {
     i=1;
}else {
     i=0;
}
```

```
i = valid ? 1 : 0;
```

ITERATION STATEMENT

- While
- Do-While
- For

WHILE

```
while(condition)
   {
   // statements to keep executing while condition is true
   ..
   ..
   }
```

Example

```
   //Increment n by 1 until n is greater than 100
   while (n > 100) {
   n = n + 1;
   }
```

DO WHILE

```
Do {
    // statements to keep executing while condition is true
} while(condition)
```

It will first executes the statement and then evaluates the condition.

Example

```
int n = 5;
Do {
System.out.println(" n = " + n);
N--;
    } while(n > 0);
```

FOR

```
for(initializer; condition; incrementer)
{
// statements to keep executing while condition is true
}
```

Example

```
int i;
int length = 10;
for (i = 0; i < length; i++) {
. . .
// do something to the (up to 9 )
. . .
}
```

JUMP STATEMENT

- Break
- Continue
- Return

JUMP STATEMENT

```java
public static void main(String args[])
{
    // Initially loop is set to run from 0-9
    for (int i = 0; i < 10; i++)
    {
        // terminate loop when i is 5.
        if (i == 5)
            break;

        System.out.println("i: " + i);
    }
    System.out.println("Loop complete.");
}
```

JUMP STATEMENT

```
public static void main(String args[])
{
    for (int i = 0; i < 10; i++)
    {
        // If the number is even
        // skip and continue
        if (i%2 == 0)
            continue;

        // If number is odd, print it
        System.out.print(i + " ");
    }
}
```

JUMP STATEMENT

```
public static void main(String args[])
{
    boolean t = true;
    System.out.println("Before the return.");

    if (t)
        return;

    // Compiler will bypass every statement
    // after return
    System.out.println("This won't execute.");
}
```

KEYWORDS

- Class
- Modifier
- Static

CLASS

- A class is a container that contains the block of code that includes field, method, constructor, etc.

```
class Person{
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public void hi() {
        System.out.println("Hi, My name is "+name);
    }
}
```

MODIFIER

- Access Modifier
  - Specify accessibility of field, method, constructor & class
- Non-Access Modifier
  - e.g. static, abstract, final

ACCESS MODIFIER

- Determine access rights for the class and its members
  - public
  - private
  - protected
  - default

ACCESS MODIFIER

Access Modifier	**Class or member can be referenced by…**
public	methods of the same class, and methods of other classes
private	methods of the same class only
protected	methods of the same class, methods of subclasses, and methods of classes in the same package
No access modifier (package access)	methods in the same package only

NON-ACCESS MODIFIER

- **Static**
- Final
- Abstract

STATIC

- Static:
  - Class
    - compile time or early binding
- Non-Static
  - Object
  - Runtime time or dynamic binding

CLASS VARIABLE
INSTANCE VARIABLE

- What is the difference between following statements?

- public int x;

- public static int x;

CLASS VARIABLE
INSTANCE VARIABLE

```
public class VariableDemo {
    static int staticVariable=0;
    int instanceVariable=0;

    public static void main(String[] args) {

        System.out.println(staticVariable); //0
        //instance variable can only be accessed through Object reference
        System.out.println(instanceVariable);

        VariableDemo object1 = new VariableDemo();
        System.out.println(object1.instanceVariable); //object reference

        object1.staticVariable = 1;
        object1.instanceVariable = 1;

        VariableDemo object2 = new VariableDemo();

        //each object has its own copy of instance variable
        System.out.println(object2.instanceVariable);

        //common to all object of a class
        System.out.println(object2.staticVariable);
```

CLASS VARIABLE
INSTANCE VARIABLE

Class Variables	**Instance Variables**
Class variables are declared with keyword *static*.	Instance variables are declared without *static* keyword.
Class variables are common to all instances of a class. These variables are shared between the objects of a class.	Instance variables are not shared between the objects of a class. Each instance will have their own copy of instance variables.
As class variables are common to all objects of a class, changes made to these variables through one object will reflect in another.	As each object will have its own copy of instance variables, changes made to these variables through one object will not reflect in another object.
Class variables can be accessed using either class name or object reference.	Instance variables can be accessed only through object reference.

STATIC METHOD
NON-STATIC METHOD

- What is the difference between following statements?

- public int getX();

- public static int getX();

STATIC METHOD
NON-STATIC METHOD

	static Method	Non-*static* Method
Access instance variables?	no	yes
Access *static* class variables?	yes	yes
Call *static* class methods?	yes	yes
Call non-*static* instance methods?	no	yes
Use the object reference *this*?	no	yes

MAIN METHOD

```
public static void main( String [] args )
{
    // application code
}
```

- main is a method
  - public — *main* can be called from outside the class
  - Static — *main* can be called by the JVM without instantiating an object
  - Void — *main* does not return a value

QUESTIONS?