JAVA FULL STACK DEVELOPMENT PROGRAM

Session 4: Java SE Basic

OUTLINE

- Variable and Primitive Data Type
- String/String Pool
- 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

```
int noOfWatts = 100; // variable declaration
```

- Declaration involves specifying the
  - Type (Data Type)
  - Name (Identifier)
  - Value (Literal) according to the type of the variable.

VARIABLE

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

VARIABLE

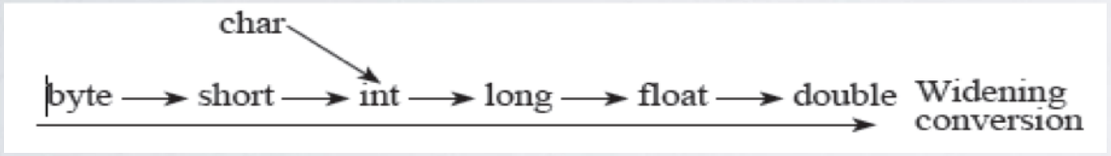
- Primitive Data Types

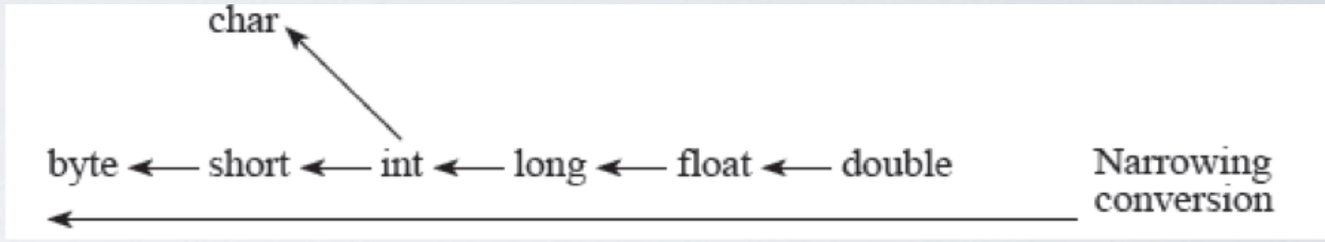
Data Type	Default Value	Size	Range
byte	0	8	–128 to 127 (inclusive)
short	0	16	–32,768 to 32,767 (inclusive)
int	0	32	–2,147,483,648 to 2,147,483,647 (inclusive)
long	0L	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive)
float	0.0F	32	1.4012984643248 17e–45f to 3.4028234766385 28860e+38f
double	0.0D	64	4.9406564584124 6544e–324 to 1.7976931348623 1570e+308
char	'\u0000'	16	0 to 65535
boolean	false	Not defined	true or false

VARIABLE

- Non-Primitive Data Types
  - String
  - Array
  - Classes
  - Interfaces
  - etc.

CONVERSION AND CASTING





- **Conversions(widening conversion) are performed automatically**
  - For e.g. a smaller box can be placed in a bigger box and so on.

- **Casting(narrowing conversion).**
  - A bigger box has to be placed in a small box.
  - Casting is not implicit in nature.
  - Use casting operator i.e. ()
  - int i = (int)(8.0/3.0);

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

IMMUTABLE CLASS

- An Immutable class in Java is declared as final
- All variables in the class is final and private
- The constructor should use deep copy to initialize all the fields
- In getter method, deep copy should be performed to return value rather than reference
- An Immutable class do not have setter methods.

STRING

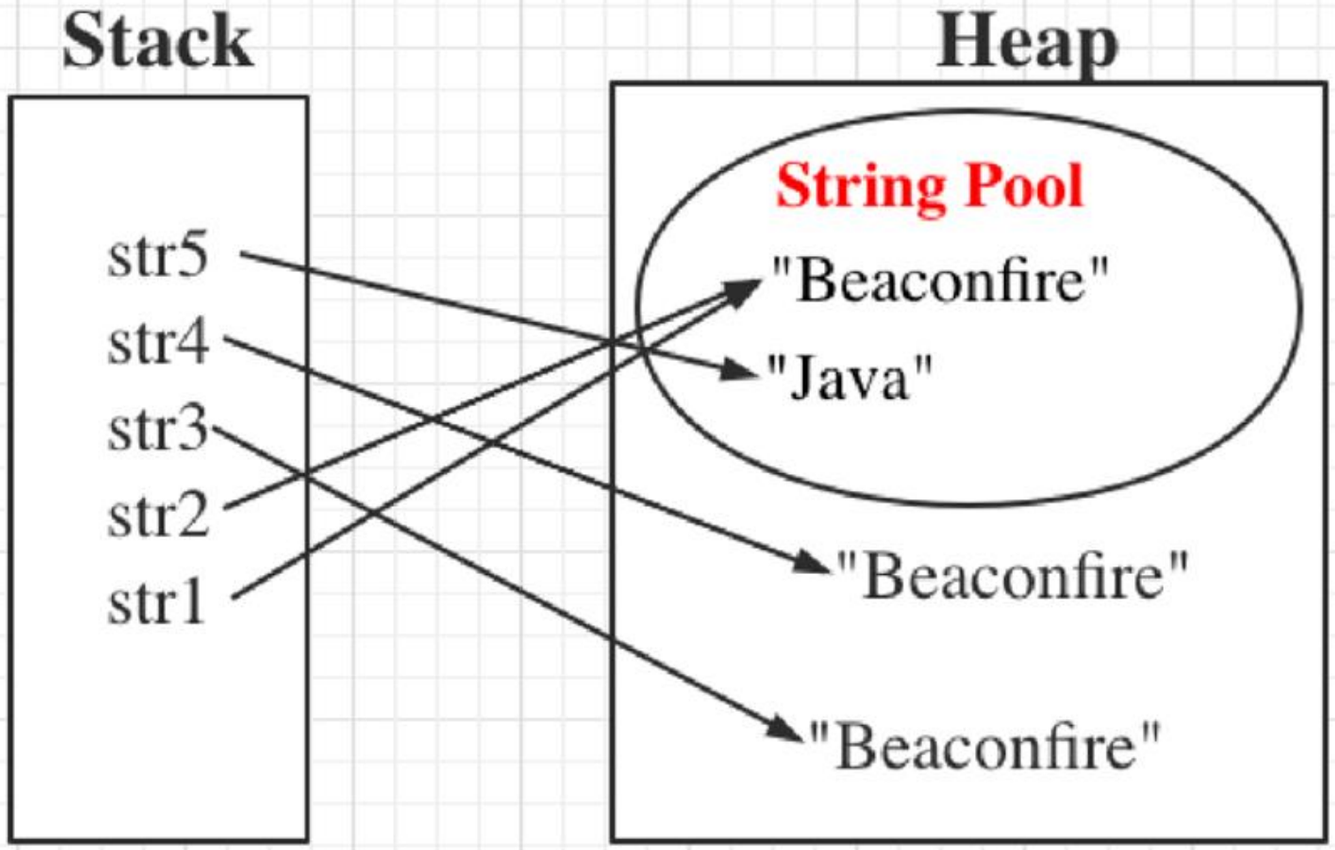
- String is a sequence of characters. In java, objects of String are **immutable** which means a constant and cannot be changed once created.

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

String Pool

```
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)

String Pool

What if I want to create a String of length 100,000 with repeating character “a”?

```
String s = "";
for (int i = 0; i < 100000; i++) {
    s += "a";
}
```

What could be a problem here?

String Builder

- String builder can boost performance when concatenating many strings together in a loop

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 100000; i++) {
    sb.append("a");
}
return sb.toString();
```

OPERATOR

- Arithmetic operations in Java
  - Precedence:

multiplicative	* / %
additive	+ -

- Parentheses: evaluate the innermost parenthesized expression first, and work your way out through the levels of nesting

- No {} or [ ] in parentheses in Java

BITWISE OPERATOR

Shift	<< >> >>>
Bitwise AND	&
Bitwise exclusive OR	^
Bitwise inclusive OR	|

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;`

Bitwise Operator can be compound operators as well

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

Name	**Operator**
Not	!
Conditional OR	||
Conditional AND	&&

RELATIONAL OPERATORS

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

FLOW CONTROL

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

SELECTION STATEMENTS

```
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.");
}
```

SELECTION STATEMENTS

- Rewrite if using ternary

```
if (a > b) {
    value = a;
} else if (b > c) {
    value = c;
} else {
    value = d;
}
```

```
value = (a > b) ? a : (b > c) ? c : d;
```

SELECTION STATEMENTS

```
switch(expression)
{
   // case statements
   // values must be of same type of expression
   case value1 :
      // Statements
      break; // break is optional

   case value2 :
      // Statements
      break; // break is optional

   // We can have any number of case statements
   // below is default statement, used when none of the cases is true.
   // No break is needed in the default case.
   default :
      // Statements
}
```

ITERATION STATEMENT

- While
- Do-While
- For

ITERATION STATEMENT

```
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;
   }
```

ITERATION STATEMENT

```
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);
```

ITERATION STATEMENT

```
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

Break terminate the loop immediately

```
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

continue skip the current iteration of the loop

```
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

return is used to return a value from a function

```
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
- Access Modifier
- Static

CLASS

- A class is a container that contains the block of code that includes field, method, constructor, etc.
- A class is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

CLASS

- Every object is an instance of a class.
- A class can contain one or more classes. This concept can be called a nested class.
- A class name must be unique within a package.

ACCESS MODIFIER

- Determine access rights for the class and its members
- Define where the class and its members can be used

FIELD MODIFIERS

- public
- private
- protected
- static
- final

FIELD 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

STATIC

- Field
- Method
- Class

STATIC FIELD/METHOD

- A static field/method belongs to the class

```
public class Demo {
    static int val = 1;
    public static void main(String[] args) {
        System.out.println(Demo.val); // 1
        Demo d = new Demo();
        System.out.println(d.val); // 1
    }
}
```

```
Integer.tostring(1223);
```

Static class

- Only nested classes can be static
- Nested static class doesn’t need reference of Outer class
- A static class cannot access non-static members of the Outer class

STATIC CLASS

```
public class CarParts {

    public static class Wheel {
        public Wheel() {
            System.out.println("Wheel created!");
        }
    }

    public CarParts() {
        System.out.println("Car Parts object created!");
    }
}
```

```
public class App {

    public static void main(String[] args) {
        CarParts.Wheel wheel = new CarParts.Wheel();
    }

}
```

STATIC METHOD VS 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

Final

- Field — constant
  - public static final MAX_VALUE = 100
- Method — prevent method overriding
  - final void show()
- Class — prevent inheritance
  - final class Demo {}

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

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.

QUESTIONS?