Basic Language Elements

Shakir Hussain

Identifiers

- A name in a program is called an Identifier.
- Used to denote classes, methods, variables and labels.

- Each character can be either `letter` or a `digits`
- `First` character in an Identifier must be a `letter` (or underscore or currency symbols like $, but never recommended)

Keywords

- Reserved words that are predefined in the language and can not be used to denote other entities.
- All keywords are in `lower case`.
- A reserved word can not be used as an Identifier.

JAVA KEYWORDS				
abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

RESERVED KEYWORDS (not currently in use)				
const	goto			

RESERVED LITERALS				
null	true	false		

Literals

- Denotes a constant value, the value that a literal represents remains unchanged in the program.

- Example :

```
Boolean literals : true, false
Integer literals :  100, 200 etc.
```

Primitive Types and Variables

- boolean, char, byte, short, int, long, float, double etc.
- These basic (or primitive) types are the only types that are not objects (due to performance issues).
- This means that you don't use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, index = 2;
double gamma = 1.2, brightness ;
boolean valueOk = false;
```

Initialisation

- If no value is assigned to local variable prior to use, then the compiler will give an error
- Java sets primitive variables to zero or false in the case of a boolean variable[non-local variable]
- All object references are initially set to null
- An array of anything is an object
  - Set to null on declaration

Declarations

```
int index = 1.2;            // compiler error
boolean retOk = 1;          // compiler error
double fiveFourths = 5 / 4;   // no error!
float ratio = 5.8f;         // correct
double fiveFourths = 5.0 / 4.0; // correct
```

- 1.2f is a float value accurate to 7 decimal places.
- 1.2 is a double value accurate to 15 decimal places.

Assignment

- All Java assignments are right associative

```
int a = 1, b = 2, c = 5;
a = b = c;
System.out.print(
"a= " + a + "b= " + b + "c= " + c)
```

- What is the value of a, b & c
- Done right to left: `a = (b = c);`

Basic Mathematical Operators

- `*` `/` `%` `+` `-` are the mathematical operators
- `*` `/` `%` have a higher precedence than `+` or `-`

```
double myVal = a + b % d - c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) -
                    ((c * d) / b);
```

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

```
name = "Fred";
```

- A block is a compound statement enclosed in curly brackets:

```
{
    name1 = "Fred"; name2 = "Bill";
}
```

- Blocks may contain other blocks

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

```
Alternation:   if, if else, switch
Looping:       for, while, do while
Escapes:       break, continue, return
```

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 10 ) x = 10;
```

- If the value of x is less than 10, make x equal to 10
- It could have been written:

```
if ( x < 10 )
x = 10;
```

- Or, alternatively:

```
if ( x < 10 ) { x = 10; }
```

Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
  System.out.print("x was changed");
}
else {
  System.out.print("x is unchanged");
}
```

Nested if … else

```
if ( myVal > 100 ) {
  if ( remainderOn == true) {
      myVal = mVal % 100;
  }
  else {
    myVal = myVal / 100.0;
  }
}
else
{
  System.out.print("myVal is in range");
}
```

else if

- Useful for choosing between alternatives:

```
if ( n == 1 ) {
  // execute code block #1
}
else if ( j == 2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed, execute
  code block #3
}
```

A Warning…

WRONG!

```
if( i == j )
    if ( j == k )

      System.out.print
      (
          "i equals
      k");
     else
      System.out.print(
       "i is not equal
       to j");
```

CORRECT!

```
if( i == j ) {
  if ( j == k )
  System.out.print(
       "i equals k");
}
else
  System.out.print("
  i is not equal to
  j");    //
  Correct!
```

The switch Statement

```
switch ( n ) {
  case 1:
    // execute code block #1
    break;
  case 2:
    // execute code block #2
    break;
    default:
    // if all previous tests fail then
    //execute code block #4
    break;
}
```

The for loop

- Loop n times

```
for ( i = 0; i < n; n++ ) {
  // this code body will execute n times
  // ifrom  0 to n-1
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {
  for ( i = 0; i < 20; i++ ){
    // this code body will execute 200 times
  }
}
```

while loops

```
while(response == 1) {
  System.out.print( "Hello World ");
  n++;
  response = readInt( "Enter response?");
}
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

do {… } while loops

```
do {
  System.out.print( "Hello World");
  n++;
  response = readInt( "Enter response?" );
}while (response == 1);
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

Break

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

```
for ( int i = 0; i < maxID, i++ ) {
  if ( userID[i] == targetID ) {
    index = i;
    break;
  }
} // program jumps here after break
```

Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```
for ( int i = 0; i < maxID; i++ ) {
  if ( userID[i] != -1 ) continue;
  System.out.print( "UserID " + i + " :" +
    userID);
}
```

Break with label

```
first:
for( int i = 0; i < 10; i++) {
  second:
  for(int j = 0; j < 5; j ++ ) {
    break myLabel;
  }
}
```

You can replace **myLabel** with first or second.

Similarly , you can have continue with label in java.


Q&A