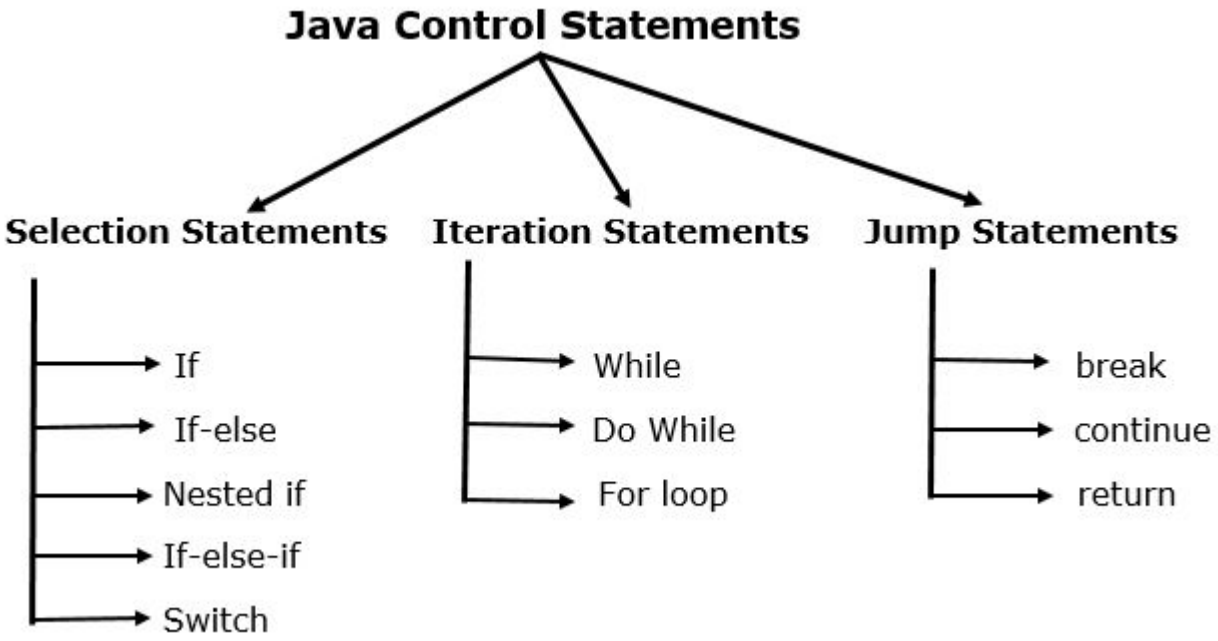
Control Statements

Control Statements



Control Statements

- If with single statement
- Nested ifs
- The if-else-if Ladder
- Switch Statement
- Nested Switch Statement (Switch statement can be used inside another switch)

Control Statements

- Iteration statement
  - While
  - do-while
  - for
- The body of the loop can be empty because a null statement (one that consists only of a semicolon) is syntactically valid in Java.

  Ex. while (++i < --j); // Midpoint of i and j

- Short loops are frequently coded without bodies when the controlling expression can handle all of the details itself.

Control Statements

- for loop

  For (initialization; condition; iteration) // iteration using loop control variable

  {

  //body

  }

- Declaring loop control variable inside loop // scope ?
- for (int n = 0; n < 10; n++) { //Code }
- Multiple initialization and iterations portions separated by comma are allowed for for loop. Ex. for (a = 1, b = 5; a < b ; a++, b--)

Control Statements

- for loop contd…
  - Either the initialization or the iteration expression or both may be absent.
  - Infinite loop using for
    - for (; ;) { }
- For-Each Version of the for Loop (also known as enhanced for loop)
- ***for(type itr-var : collection) statement-block***

```
int nums [ ] = {1,2,3,4,5};
int sum = 0;
for (int x: nums) sum += x;
```

- Its iteration variable is read only i.e. x = x + 2 will not affect nums.
- The iteration variable in the for loop must be type compatible with the type of array being obtained. // Accessing double dimension array

Control Statements

Jump Statement

- break - switch, loop, “civilized form of goto” i.e. labeled break
- continue - loop, labelled continue
- return - explicitly return the control back to the caller
- Void fun()

```
{
        // statement1;
        return;
        Statement2;          //”Unreachable code” error - what is the solution?
}
```

Control Statements

Jump Statement

- We can assign a label to a block of code.
- We can not break to any label which is not defined for an enclosing block.

Ex.

```
first: {
        second: {
                break first;            // Control will come out of the first block
        }
        // break second;                // Error, this statement must be inside second block.
}
```

Control Statement: Labeled Continue

```
class LabeledContinue {
    public static void main(String args[]) {
        outer: for (int i = 1; i <= 5; i++)
        {
            inner: for (int j = 1; j <= 5; j++)
            {
                if(j > i) {
                    System.out.println();
                    continue outer;
                }
                else System.out.print(j);
            }
        }
    }
}
```

// GUESS OUTPUT ??

Control Statement: Labeled break

```
class LabeledContinue {
    public static void main(String args[]) {
        outer: for (int i = 1; i <= 5; i++)
        {
            inner: for (int j = 1; j <= 5; j++)
            {
                if(j > i) {
                    System.out.println();
                    break outer;
                }
                else System.out.print(j);
            }
        }
    }
}
```

// GUESS OUTPUT ??

```
class SwitchCase{
	public static void main(String args[]){
		int choice = 4;
		switch(choice)
		    {
			default:
				System.out.println("default");
				break; //This break is important here, otherwise next case will continue.
			case 1:
				System.out.println("Case 1");
				break;
			case 2:
				System.out.println("Case 2");
				break;
		    }
		System.out.println("Out of switch-case");
	}
}
```

GUESS OUTPUT ??