Control Statements

Control statement summary

- Selection statements
  - if
  - if ...else
  - Nested if
  - if-else-if
  - switch & nested switch
- Looping statements
  - while
  - do ...while
  - for
  - break
  - continue
  - return

if statements

i) if(condition)
    statement1;

ii) if(condition)
    Statement1;
  else
    Statement2;

iii) if(condition) {
    Stmt1;
    Stmt2;
    ...
  }

iv) if(condition) {
    Stmt1;
    Stmt2;
    ...
  }
  else
    Stmtk;

ii) if(condition)
    Statement1;
  else{
    Stmt2;
    Stmt3;
    ...
  }

v) if(condition) {
    Stmt1;
    Stmt2;
    ...
  }
  else {
    Stmtk;
    Stmtk+1;
    ...
  }

if-else-if example

```
class IfElse {
  public static void main(String args[]) {
    int month = 4; // April
    String season;

    if(month == 12 || month == 1 || month == 2)
      season = "Winter";
    else if(month == 3 || month == 4 || month == 5)
      season = "Spring";
    else if(month == 6 || month == 7 || month == 8)
      season = "Summer";
    else if(month == 9 || month == 10 || month == 11)
      season = "Autumn";
    else
      season = "Bogus Month";

    System.out.println("April is in the " + season + ".");
  }
}
```

switch

```
switch (expression) {
    case value1:
        // statement sequence
        break;
    case value2:
        // statement sequence
        break;
        .
        .
        .
    case valueN:
        // statement sequence
        break;
    default:
        // default statement sequence
}
```

```
class SampleSwitch {
  public static void main(String args[]) {
    for(int i=0; i<6; i++)
      switch(i) {
        case 0:
          System.out.println("i is zero.");
          break;
        case 1:
          System.out.println("i is one.");
          break;
        case 2:
          System.out.println("i is two.");
          break;
        case 3:
          System.out.println("i is three.");
          break;
        default:
          System.out.println("i is greater than 3.");
      }
  }
}
```

loops

- while

```
while(condition) {
    // body of loop
}
```

- do-while

```
do {
    // body of loop
} while (condition);
```

- for

```
for(initialization; condition; iteration)
{
    // body
}
```

- For-Each Version

```
for(type itr-var : collection)
        statement-block
```

Example: for each

```
// Use a for-each style for loop.
class ForEach {
	public static void main(String args[]) {
		int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
		int sum = 0;
		// use for-each style for to display and sum the values
		for(int x : nums) {
			System.out.println("Value is: " + x);
			sum += x;
		}
		System.out.println("Summation: " + sum);
	}
}
```

break

- break statement has three uses:
  - It terminates a statement sequence in a switch statement.
  - It can be used to exit a loop.
  - It can be used as a “civilized” form of goto

break in the form of goto

- By using this form of break, we can break out of one or more blocks of code.
- These blocks need not be part of a loop or a switch. They can be any block.
- Further, we can specify precisely where execution will resume, because this form of break works with a label.
  - break label;

example

```
// Using break as a civilized form of goto.
class Break {
  public static void main(String args[]) {
    boolean t = true;

    first: {
      second: {
        third: {
          System.out.println("Before the break.");
          if(t) break second; // break out of second block
          System.out.println("This won't execute");
        }
        System.out.println("This won't execute");
      }
      System.out.println("This is after second block.");
    }
  }
}
```

continue

- In while and do-while loops, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.
- In a for loop, control goes first to the iteration portion of the for statement and then to the conditional expression.
- For all three loops, any intermediate code is bypassed.

return

- The return statement is used to explicitly return from a method.
- It is categorized as a jump statement.
- At any time in a method the return statement can be used to cause execution to branch back to the caller of the method.
- Thus, the return statement immediately terminates the method in which it is executed.

```
class Return {
  public static void main(String args[])
{
    boolean t = true;

    System.out.println("Before the
return.");

    if(t) return; // return to caller

    System.out.println("This won't
execute.");
  }
}
```