Control Structures

Control Structures

- Control structures
  - allows us to change the ordering of how the statements in our programs are executed
- Two types of Control Structures
  - decision control structures
    - allows us to select specific sections of code to be executed
  - repetition control structures
    - allows us to execute specific sections of the code a number of times

Decision Control Structures

- Decision control structures
  - Java statements that allow us to select and execute specific blocks of code while skipping other sections
- Types:
  - if-statement
  - if-else-statement
  - If-else if-statement
  - switch

if-statement

- if-statement
  - specifies that a statement (or block of code) will be executed if and only if a certain boolean statement is true.

- if-statement has the form:

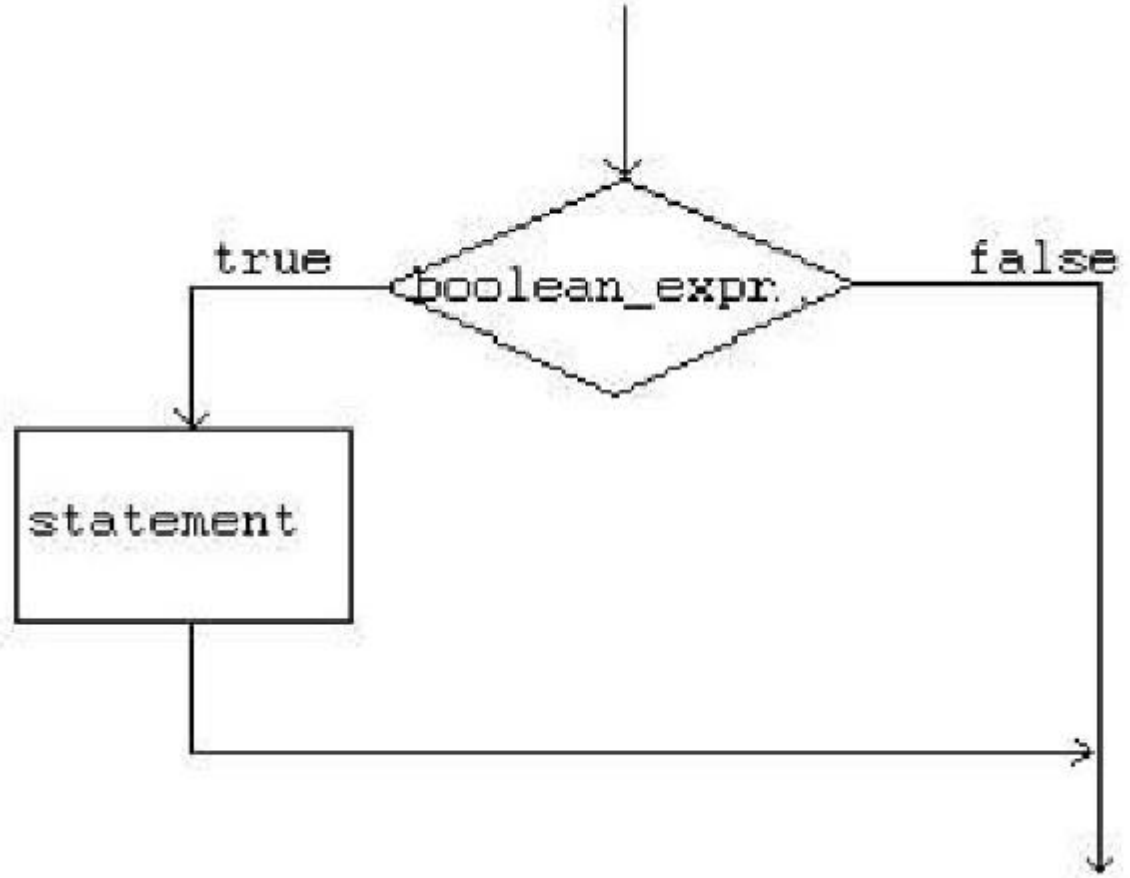
```
if(  boolean_expression  )
      statement;
```

or

```
if(  boolean_expression  ){
      statement1;
      statement2;
}
```

  - where,
    - boolean_expression is either a boolean expression or boolean variable.

if-statement Flowchart



Example 1

```
int grade = 68;
if( grade > 60 )
   System.out.println("Congratulations!");
```

Example 2

```
int grade = 68;
if( grade > 60 ){
   System.out.println("Congratulations!");
   System.out.println("You passed!");
}
```

Coding Guidelines

1. The **boolean_expression** part of a statement should evaluate to a boolean value. That means that the execution of the condition should either result to a value of true or a false.
2. Indent the statements inside the if-block.

   For example,

```
if(   boolean_expression   ){
        //statement1;
        //statement2;

}
```

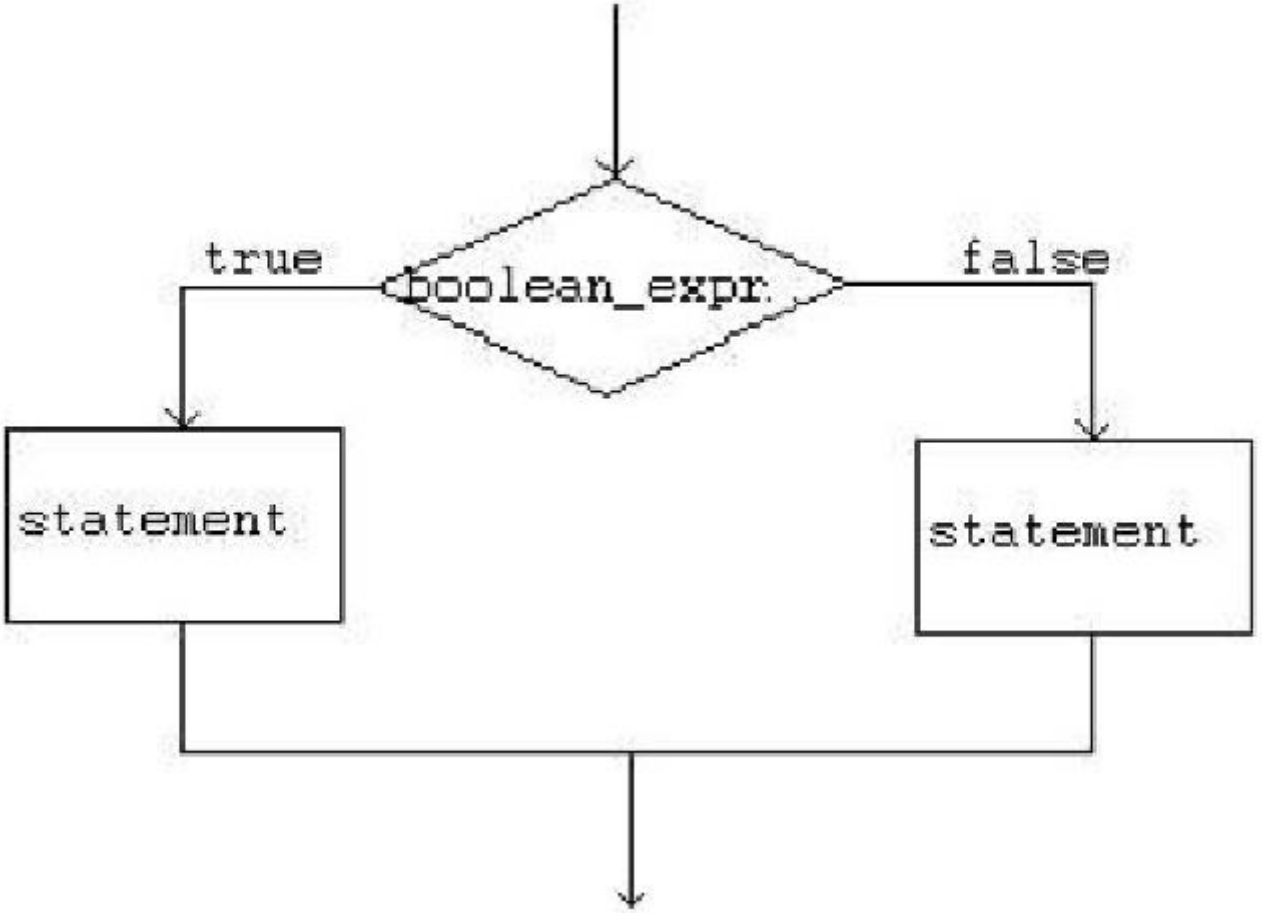
if-else statement

- if-else statement
  - used when we want to execute a certain statement if a condition is true, and a different statement if the condition is false.

- if-else statement has the form:

```
if(  boolean_expression  ){
      statement1;
      statement2;
      .  .  .
}
else{
      statement3;
      statement4;
      .  .  .
}
```

Flowchart



Example 1

```
int grade = 68;


if( grade > 60 )
   System.out.println("Congratulations!");
else
   System.out.println("Sorry you failed");
```

Example 2

```
int grade = 68;

if( grade > 60 ){
   System.out.println("Congratulations!");
   System.out.println("You passed!");
}
else{
   System.out.println("Sorry you failed");
}
```

Coding Guidelines

1. To avoid confusion, always place the statement or statements of an if or if-else block inside brackets {}.
2. You can have nested if-else blocks. This means that you can have other if-else blocks inside another if-else block.

   For example,

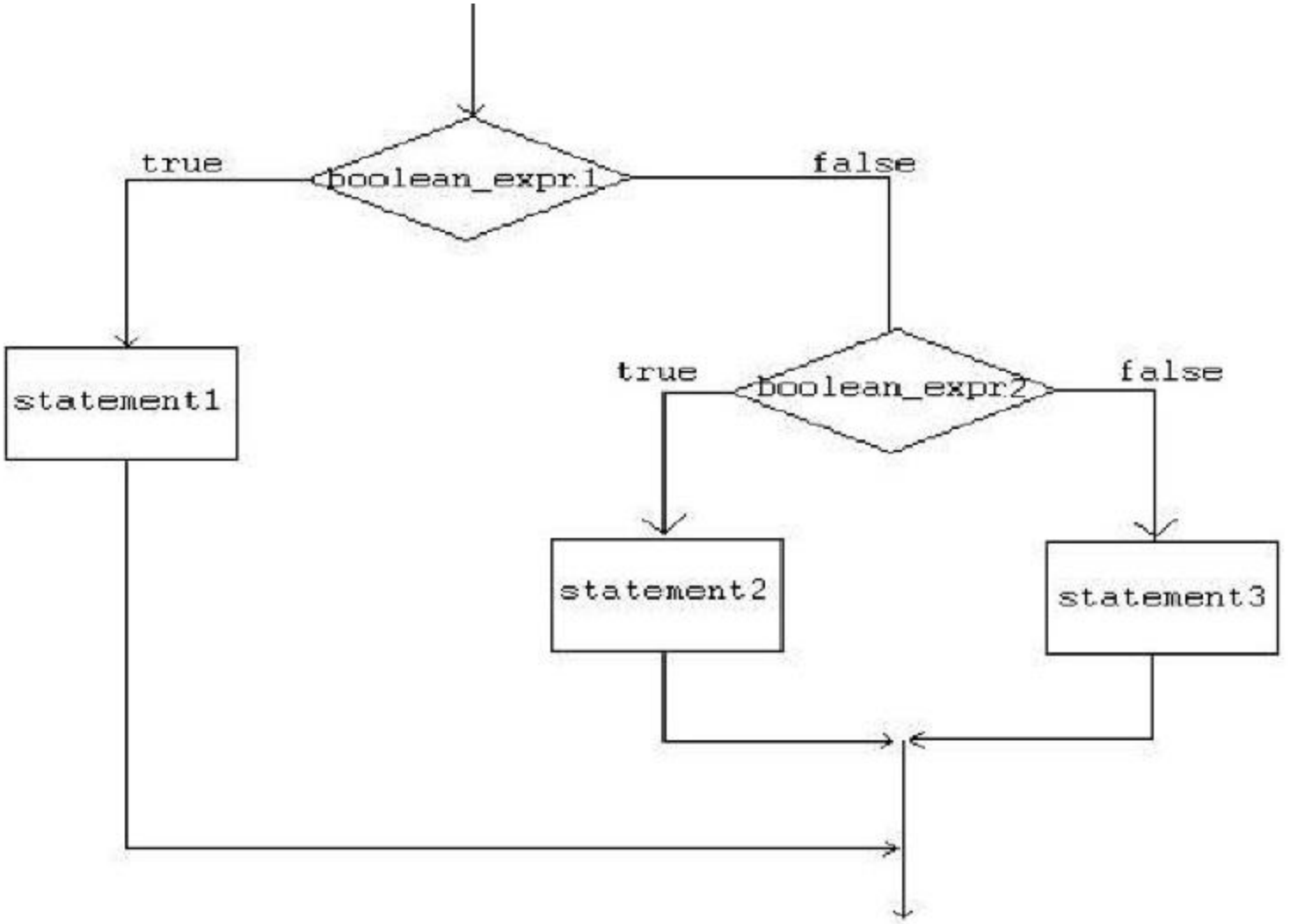
```
if(  boolean_expression   ){
    if(   boolean_expression  ){
          //some   statements  here
    }

}

else{
          //some   statements  here

}
```

if-else-else if statement

- The statement in the else-clause of an if-else block can be another if-else structures.
- This cascading of structures allows us to make more complex selections.
- The statement has the form:

```
if(  boolean_expression1   )
      statement1;
else   if(   boolean_expression2   )
      statement2;
else
      statement3;
```

Flowchart


true
boolean_expr1
false
statement1
true
boolean_expr2
false
statement2
statement3

Example

```
int   grade   =   68;

if(   grade   >   90   ){
      System.out.println("Very   good!");
}
else   if(   grade   >   60   ){
      System.out.println("Very   good!");
}
else{
      System.out.println("Sorry   you   failed");
}
```

Common Errors

1. The condition inside the if-statement does not evaluate to a boolean value. For example,

```
//WRONG
int  number   =  0;
if(  number   ){
      //some  statements   here
}
```

The variable number does not hold a boolean value.

2. Writing elseif instead of else if.

Common Errors

3. Using = instead of == for comparison.

For example,

```
//WRONG
int  number   =  0;
if(  number  =   0   ){
      //some  statements   here
}
```

This should be written as,

```
//CORRECT
int  number   =  0;
if(  number  ==   0   ){
      //some  statements   here
}
```

Sample Program

```
1    public class Grade    {
2          public static void main( String[] args )
3          {
4                double grade = 92.0;
5                if( grade >= 90 ){
6                      System.out.println( "Excellent!" );
7                }
8                else if( (grade < 90) && (grade >= 80)){
9                      System.out.println("Good job!" );
10               }
11               else if( (grade < 80) && (grade >= 60)){
12                     System.out.println("Study harder!" );
13               }
14               else{
15                     System.out.println("Sorry, you failed.");


16                 }
17           }
18     }
```

switch-statement

- switch
  - allows branching on multiple outcomes.
- switch statement has the form:

```
switch(   switch_expression              ){
      case  case_selector1:
            statement1;//
            statement2;//block           1
            break;
      case  case_selector2:
            statement1;//
            statement2;//block           2
            break;
                  :
      default:
            statement1;//
            statement2;//block           n
}
```

switch-statement

- where,
  - switch_expression
    - is an integer or character expression
  - case_selector1, case_selector2 and so on,
    - are unique integer or character constants.

From Java 7 version, ‘switch’ allows Strings

- This is better than using multiple if-else statements
- Better performace
- Readability

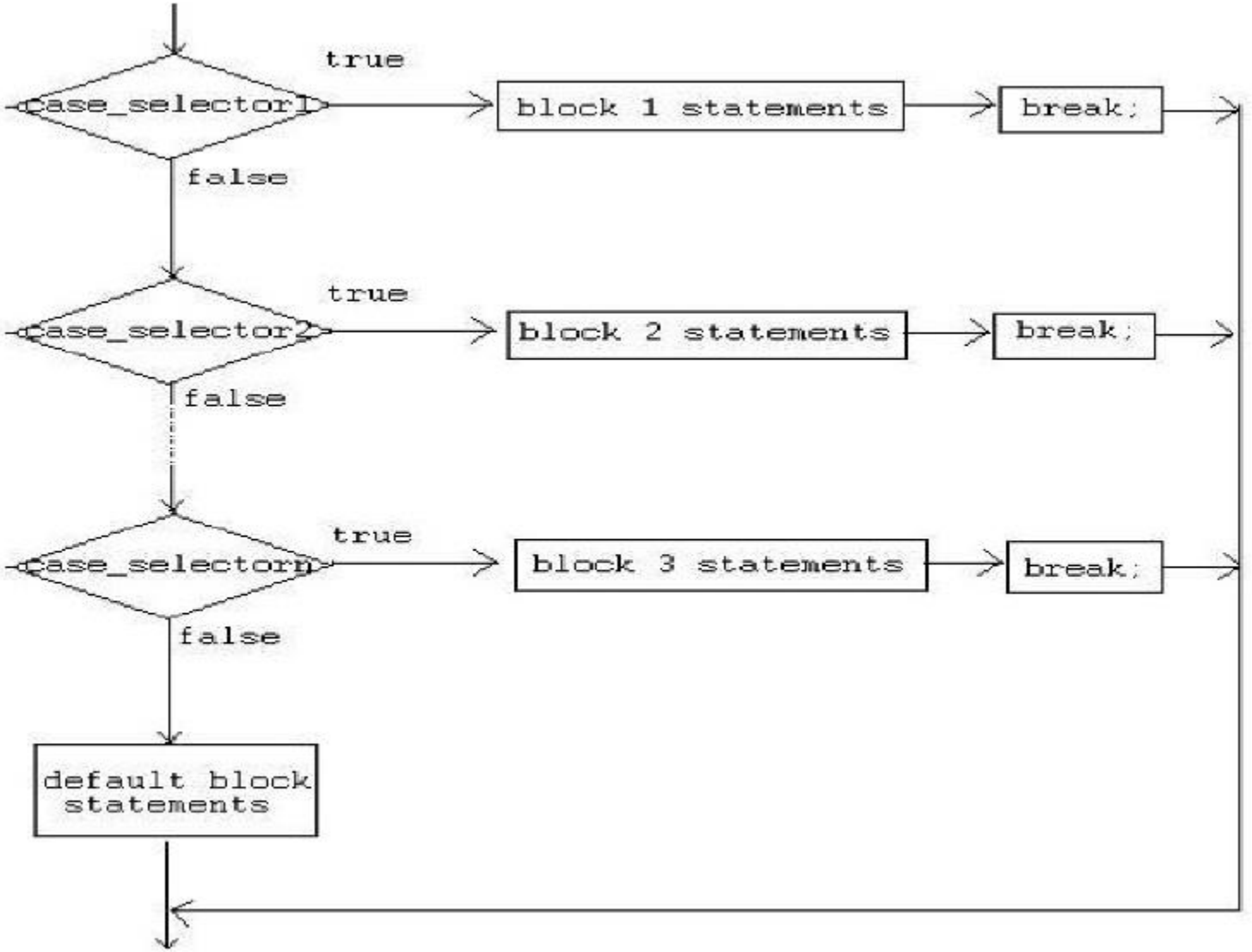
switch-statement

- When a switch is encountered,
  - Java first evaluates the switch_expression, and jumps to the case whose selector matches the value of the expression.
  - The program executes the statements in order from that point on until a break statement is encountered, skipping then to the first statement after the end of the switch structure.
  - If none of the cases are satisfied, the default block is executed. Take note however, that the default part is optional.

switch-statement

- NOTE:
  - Unlike with the if statement, the multiple statements are executed in the switch statement without needing the curly braces.
  - When a case in a switch statement has been matched, all the statements associated with that case are executed. Not only that, the statements associated with the succeeding cases are also executed.
  - To prevent the program from executing statements in the subsequent cases, we use a break statement as our last statement.

Flowchart


true
case_selector1
block 1 statements
break;
false
true
case_selector2
block 2 statements
break;
false
true
case_selectorn
block 3 statements
break;
false
default block
statements

Example

```
1   public class Grade {
2           public static void main( String[] args )
3           {
4                   int grade = 90;
5                   switch(grade){
6                       case 100:
7                           System.out.println( "Excellent!" );
8                             break;
9                     case 90:
10                          System.out.println("Good job!" );
11                          break;
12                      case 80:
13                          System.out.println("Study harder!" );
14                          break;
15                      default:
16                          System.out.println("Sorry, you failed.");
17                  }
18            }
19  }
```

Coding Guidelines

1. Deciding whether to use an if statement or a switch statement is a judgment call. You can decide which to use, based on readability and other factors.
2. An if statement can be used to make decisions based on ranges of values or conditions, whereas a switch statement can make decisions based only on a single integer or character value. Also, the value provided to each case statement must be unique.

Repetition Control Structures

- Repetition control structures
  - are Java statements that allows us to execute specific blocks of code a number of times.

- Types:
  - while-loop
  - do-while loop
  - for-loop

while-loop

- while loop
  - is a statement or block of statements that is repeated as long as some condition is satisfied.

- while loop has the form:

```
while(  boolean_expression  ){
        statement1;
        statement2;
        .  .  .
}
```

  - The statements inside the while loop are executed as long as the boolean_expression evaluates to true.

Example 1

```
int   x   =   0;

while   (x<10)          {
        System.out.println(x);
        x++;
}
```

Example 2

```
//infinite   loop
while(true)
      System.out.println(“hello”);
```

Example 3

```
//no   loops
//  statement   is   not   even   executed
while   (false)
        System.out.println(“hello”);
```

do-while-loop

- do-while loop
  - is similar to the while-loop
  - statements inside a do-while loop are executed several times as long as the condition is satisfied
  - The main difference between a while and do-while loop:
    - the statements inside a do-while loop are executed at least once.

- do-while loop has the form:

```
do{
    statement1;
    statement2;
    . . .
}while(  boolean_expression  );
```

Example 1

```
int   x   =   0;

do     {
       System.out.println(x);
       x++;
}while   (x<10);
```

Example 2

```
//infinite   loop
do{
      System.out.println(“hello”);
}   while   (true);
```

Example 3

```
//one   loop
//   statement   is   executed   once
do
      System.out.println(“hello”);
while   (false);
```

Coding Guidelines

1. Common programming mistakes when using the do-while loop is forgetting to write the semi-colon after the while expression.

```
do{
    ...
 }while(boolean_expression)//WRONG->forgot semicolon;
```

2. Just like in while loops, make sure that your do-while loops will terminate at some point

for-loop

- for loop
  - allows execution of the same code a number of times.

- for loop has the form:

```
for(InitializationExpression; LoopCondition; StepExpression)
{
        statement1;
        statement2;
        .   .   .
}
```

  - where,
    - InitializationExpression -initializes the loop variable.
    - LoopCondition - compares the loop variable to some limit value.
    - StepExpression - updates the loop variable.

Example

```
int   i;
for(   i   =   0;   i   <   10;   i++   ){
      System.out.println(i);
}
```

- The code shown above is equivalent to the following while loop.

```
int   i   =   0;
while(   i   <   10   ){
      System.out.print(i);
      i++;
}
```

Branching Statements

- Branching statements allows us to redirect the flow of program execution.

- Java offers three branching statements:
  - break
  - continue
  - return.

Unlabeled break statement

- unlabeled break
  - terminates the enclosing switch statement, and flow of control transfers to the statement immediately following the switch.
  - This can also be used to terminate a for, while, or do-while loop

Example

```
String names[]={"Beah","Bianca","Lance","Belle","Nico","Yza","Gem","Ethan"};


String    searchName = "Yza";
boolean   foundName = false;


for( int i=0; i< names.length; i++ ){
    if( names[i].equals( searchName )){
        foundName = true;
        break;
    }
}
if( foundName )   System.out.println( searchName + " found!" );
else              System.out.println( searchName + " not found." );
```

labeled break statement

- labeled break statement
  - terminates an outer statement, which is identified by the label specified in the break statement.
  - the flow of control transfers to the statement immediately following the labeled (terminated) statement.
  - The sample program in the next slide searches for a value in a two-dimensional array. Two nested for loops traverse the array. When the value is found, a labeled break terminates the statement labeled search, which is the outer for loop.

Example

```
int[][] numbers = {{1, 2, 3}, {4, 5, 6},{7, 8, 9}};
int searchNum = 5;
boolean foundNum = false;

searchLabel:
for( int i=0; i<numbers.length; i++ ){
    for( int j=0; j<numbers[i].length; j++ ){
        if( searchNum == numbers[i][j] ){
            foundNum = true;
            break searchLabel;
        }
    }
}
if( foundNum )   System.out.println(searchNum + " found!" );
else             System.out.println(searchNum + " not found!");
```

Unlabeled continue statement

- unlabeled continue statement
  - skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop, basically skipping the remainder of this iteration of the loop.

Example

```
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int   count = 0;

for( int i=0; i<names.length; i++ ){
    if( !names[i].equals("Beah") ){
        continue;   //skip next statement
    }
    count++;
}
System.out.println("There are "+count+" Beahs in the list");
```

Labeled continue statement

- labeled continue statement
  - skips the current iteration of an outer loop marked with the given label.

Example

```
outerLoop:
for( int i=0; i<5; i++ ){
    for( int j=0; j<5; j++ ){
        System.out.println("Inside for(j) loop"); //message1
        if( j == 2 )  continue outerLoop;
    }
    System.out.println("Inside for(i) loop"); //message2
}
```

- In this example, message 2 never gets printed since we have the statement continue outerloop which skips the iteration.

return statement

- return statement
  - used to exit from the current method.
  - flow of control returns to the statement that follows the original method call.

return statement

- To return a value
  - simply put the value (or an expression that calculates the value) after the return keyword.
  - For example,

```
return ++count;
```

or

```
return   "Hello";
```

  - The data type of the value returned by return must match the type of the method's declared return value.

return statement

- When a method is declared void, use the form of return that doesn't return a value.
  - For example,

```
return;
```

- We will cover more about return statements later when we discuss about methods.

Summary

- Decision Control Structures
  - if
  - if-else
  - if – else if
  - switch
- Repetition Control Structures
  - while
  - do-while
  - for
- Branching Statements
  - break
  - continue
  - return