The creation of software involves four basic activities:
- establishing the requirements
- creating a design
- implementing the code
- testing the implementation

The development process is much more involved than this, but these are the four basic development activities.

Software requirements specify the tasks a program must accomplish (what to do, not how to do it).

A software design specifies how a program will accomplish its requirements:
- In object-oriented development, the design establishes the classes, objects, methods, and data that are required.
- Implementation is the process of translating a design into source code.
- Almost all important decisions are made during requirements and design stages.

A program should be executed multiple times with various input in an attempt to find errors.

Debugging is the process of discovering the causes of problems and fixing them.

A conditional statement lets us choose which statement will be executed next.

Therefore they are sometimes called selection statements.

Conditional statements give us the power to make basic decisions.

Java’s conditional statements are:
- the if statement
- the if-else statement
- the switch statement

The if statement

The if statement has the following syntax:

```java
if (condition) statement;
```

The condition must be a boolean expression. It must evaluate to either true or false.

- if is a Java reserved word.
- The statement must be a valid Java statement.
- If the condition is true, the statement is executed.
- If it is false, the statement is skipped.

A condition often uses one of Java’s equality operators or relational operators, which all return boolean results:

- `==` equal to
- `!=` not equal to
- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to

Note the difference between the equality operator (==) and the assignment operator (=).
The if-else Statement

- An else clause can be added to an if statement to make an if-else statement

```java
if (condition)
    statement1;
else
    statement2;
```

- If the condition is true, statement1 is executed; if the condition is false, statement2 is executed.

- One or the other will be executed, but not both.

---

Block Statements

- Several statements can be grouped together into a block statement.
- A block is delimited by braces: `{ ... }`
- A block statement can be used wherever a statement is called for by the Java syntax.
- For example, in an if-else statement, the if portion, or the else portion, or both, could be block statements.

---

Logical Operators

- Boolean expressions can use the following logical operators:
  - Logical NOT
  - Logical AND
  - Logical OR

- They all take boolean operands and produce boolean results.
- Logical NOT is a unary operator (it operates on one operand).
- Logical AND and logical OR are binary operators (each operates on two operands).

---

Example

```java
import ca.keyboard;
public class Wages
{
    //Reads the number of hours worked and calculates wages.
    public static void main(String[] args)
    {
        final double RATE = 8.25;  // regular pay rate
        final int STANDARD = 40;   // standard hours in a work week
        double pay = 0.0;
        System.out.print("Enter the number of hours worked: ");
        int hours = Keyboard.readInt();
        System.out.println();
        // Pay overtime at "time and a half"
        if (hours > STANDARD)
            pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
        else
            pay = hours * RATE;
        System.out.println("Gross earnings: "+ pay);
    }
}
```
Short Circuited Operators
- The processing of logical AND and logical OR is “short-circuited”
- If the left operand is sufficient to determine the result, the right operand is not evaluated
  ```java
  if (count != 0 && total/count > MAX)
  System.out.println("Testing.");
  ```
- This type of processing must be used carefully

Comparing Strings
- Remember that a character string in Java is an object
- We cannot use the relational operators to compare strings
- The equals method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The String class also contains a method called compareTo to determine if one string comes before another (based on the Unicode character set)

Lexicographic Ordering
- Because comparing characters and strings is based on a character set, it is called a lexicographic ordering
- This is not strictly alphabetical when uppercase and lowercase characters are mixed
- For example, the string "Great" comes before the string "fantastic" because all of the uppercase letters come before all of the lowercase letters in Unicode
- Also, short strings come before longer strings with the same prefix (lexicographically)
- Therefore "book" comes before "bookcase"

Comparing Float Values
- We also have to be careful when comparing two floating point values (float or double) for equality
- You should rarely use the equality operator (==) when comparing two floats
- In many situations, you might consider two floating point numbers to be “close enough” even if they aren’t exactly equal
- Therefore, to determine the equality of two floats, you may want to use the following technique:
  ```java
  if (Math.abs(f1 - f2) < 0.00001)
  System.out.println("Essentially equal.");
  ```

More Operators
- To round out our knowledge of Java operators, let’s examine a few more
- In particular, we will examine
  - the increment and decrement operators
  - the assignment operators

Increment and Decrement
- The increment and decrement operators are arithmetic and operate on one operand
- The increment operator (++) adds one to its operand
- The decrement operator (--) subtracts one from its operand
- The statement
  ```java
  count++;
  ```
  is functionally equivalent to
  ```java
  count = count + 1;
  ```
Increment and Decrement

- The increment and decrement operators can be applied in prefix form (before the operand) or postfix form (after the operand).
- When used alone in a statement, the prefix and postfix forms are functionally equivalent. That is,
  \[
  \text{count}++; \quad \text{is equivalent to} \quad ++\text{count};
  \]

Increment and Decrement

- When used in a larger expression, the prefix and postfix forms have different effects.
- In both cases the variable is incremented (decremented).
- But the value used in the larger expression depends on the form used:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Operation</th>
<th>Value Used in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>count++</td>
<td>add 1</td>
<td>old value</td>
</tr>
<tr>
<td>++count</td>
<td>add 1</td>
<td>new value</td>
</tr>
<tr>
<td>count--</td>
<td>subtract 1</td>
<td>old value</td>
</tr>
<tr>
<td>--count</td>
<td>subtract 1</td>
<td>new value</td>
</tr>
</tbody>
</table>

Assignment Operators

- There are many assignment operators, including the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += y</td>
<td>x = x + y</td>
</tr>
<tr>
<td>-=</td>
<td>x -= y</td>
<td>x = x - y</td>
</tr>
<tr>
<td>*=</td>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
</tr>
<tr>
<td>%=</td>
<td>x %= y</td>
<td>x = x % y</td>
</tr>
</tbody>
</table>

Assignment Operators

- The behavior of some assignment operators depends on the types of the operands.
- If the operands to the += operator are strings, the assignment operator performs string concatenation.
- The behavior of an assignment operator (+=) is always consistent with the behavior of the "regular" operator (+).

Repetition Statements

- Repetition statements allow us to execute a statement multiple times.
- Often they are referred to as loops.
- Like conditional statements, they are controlled by boolean expressions.
- Java has three kinds of repetition statements:
  - the while loop
  - the do loop
  - the for loop
- The programmer should choose the right kind of loop for the situation.

The while Statement

- The while statement has the following syntax:

\[
\text{while (condition)} \\
\text{statement;}
\]
- If the condition is true, the statement is executed. Then the condition is evaluated again.
- The statement is executed repeatedly until the condition becomes false.
Example

//**********************************************************
//  Counter.java Author: Lewis/Loftus
//  Demonstrates the use of a while loop.
//**********************************************************
public class Counter
{
    //Prints integer values from 1 to a specific limit.
    public static void main(String[] args)
    {
        final int LIMIT = 5;
        int count = 1;
        while (count <= LIMIT)
        {
            System.out.println(count);
            count = count + 1;
        }
        System.out.println("Done");
    }
}

Example

import cs1.Keyboard;
public class WinPercentage
{
    //Computes the percentage of games won by a team.
    public static void main(String[] args)
    {
        final int NUM_GAMES = 12;
        int won;
        double ratio;
        System.out.print("Enter the number of games won (0 to "+ NUM_GAMES + "): ");
        won = Keyboard.readInt();
        while (won < 0 || won > NUM_GAMES)
        {
            System.out.print("Invalid input. Please reenter: ");
            won = Keyboard.readInt();
        }
        ratio = (double) won / NUM_GAMES;
        System.out.println();
        System.out.println("Winning percentage: " + ratio);
    }
}

Infinite Loops

- The body of a while loop eventually must make the condition false
- If not, it is an infinite loop, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check to ensure that your loops will terminate normally

Example

//**********************************************************
//  Forever.java Author: Lewis/Loftus
//  Demonstrates an INFINITE LOOP. WARNING!!
//**********************************************************
public class Forever
{
    //Prints ever decreasing integers in an INFINITE LOOP!
    public static void main(String[] args)
    {
        int count = 1;
        while (count <= 25)
        {
            System.out.println(count);
            count = count - 1;
        }
        System.out.println("Done"); //this statement never reached
    }
}

The do Statement

- The do statement has the following syntax:
  do
  {
    statement;
  }
  while (condition)

  The statement is executed once initially, and then the condition is evaluated.
  The statement is executed repeatedly until the condition becomes false.

Example

//**********************************************************
//  do-While.java Author: Lewis/Loftus
//  Demonstrates the use of a do-while loop.
//**********************************************************
public class do_While
{
    //Prints ever decreasing integers in a do-while loop.
    public static void main(String[] args)
    {
        int count = 1;
        do
        {
            System.out.println(count);
            count = count - 1;
        }
        while (count <= 25);
        System.out.println("Done");
    }
}

The do Statement

- A do loop is similar to a while loop, except that the condition is evaluated after the body of the loop is executed
- Therefore the body of a do loop will execute at least once
Example

```java
//**********************************************************
//  Counter2.java       Author: Lewis/Loftus
//  Demonstrates the use of a do loop.
//**********************************************************
public class Counter2 {
    //Prints integer values from 1 to a specific limit.
    public static void main (String[] args) {
        final int LIMIT = 5;
        int count = 0;
        do {
            count = count + 1;
            System.out.println (count);
        } while (count < LIMIT);
        System.out.println ("Done");
    }
}
```

Example

```java
import cs1.Keyboard;
public class ReverseNumber {
    //Reverses the digits of an integer mathematically.
    public static void main (String[] args) {
        int number, lastDigit, reverse = 0;
        System.out.print ("Enter a positive integer: ");
        number = Keyboard.readInt();
        do {
            lastDigit = number % 10;
            reverse = (reverse * 10) + lastDigit;
            number = number / 10;
        } while (number > 0);
        System.out.println ("That number reversed is " + reverse);
    }
}
```

Comparing while and do

- while loop:
  - **true** condition evaluated
  - statement
  - false condition-statement-increment cycle

- do loop:
  - **true** condition
  - statement
  - false condition-statement-increment cycle

The for Statement

- The for statement has the following syntax:
  - for (initialization; condition; increment)
  - The initialization is executed once before the loop begins.
  - The statement is executed until the condition becomes false.
  - The increment portion is executed at the end of each iteration.
  - The condition-statement-increment cycle is executed repeatedly.

The for Statement

- A for loop is functionally equivalent to the following while loop structure:
  ```java
  initialization;
  while (condition)
  {
    statement;
    increment;
  }
  ```

Logic of a for loop

- Initialization
  - true
  - false

- Condition
  - true
  - false

- Statement
  - false

- Increment
  - false
The for Statement

- Like a while loop, the condition of a for statement is tested prior to executing the loop body.
- Therefore, the body of a for loop will execute zero or more times.
- It is well suited for executing a loop a specific number of times that can be determined in advance.

Example

```java
import cs1.Keyboard;
public class Multiples {
    // Prints multiples of a user-specified number up to a user-specified limit.
    public static void main(String[] args) {
        final int PER_LINE = 5;
        int value, limit, mult, count = 0;
        System.out.print("Enter a positive value: ");
        value = Keyboard.readInt();
        System.out.print("Enter an upper limit: ");
        limit = Keyboard.readInt();
        System.out.println();
        System.out.println("The multiples of " + value + " between " + value + " and " + limit + " (inclusive) are:");
        for (mult = value; mult <= limit; mult += value) {
            System.out.print(mult + "	");
            // Print a specific number of values per line of output
            count++;
            if (count % PER_LINE == 0) {
                System.out.println();
            }
        }
    }
}
```

Choosing a Loop Structure

- When you can't determine how many times you want to execute the loop body, use a while statement or a do statement.
  - If it might be zero or more times, use a while statement.
  - If it will be at least once, use a do statement.
- If you can determine how many times you want to execute the loop body, use a for statement.

The for Statement

- Each expression in the header of a for loop is optional.
  - If the initialization is left out, no initialization is performed.
  - If the condition is left out, it is always considered to be true, and therefore creates an infinite loop.
  - If the increment is left out, no increment operation is performed.
  - Both semi-colons are always required in the for loop header.