COSC 123
Computer Creativity

Java Decisions and Loops

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) A decision is made by evaluating a condition in an if/else statement and performing certain actions depending if the condition is true or false.

2) Repetition is performed by using loops that repeat a set of statements multiple times.

Making Decisions

Decisions are used to allow the program to perform different actions in certain conditions.

- For example, if a person applies for a driver's license and is not 16, then the computer should not give them a license.

To make a decision in a program we must do several things:

- 1) Determine the ***condition*** in which to make the decision.
  - ⇨ In the license example, we will not give a license if the person is under 16.
- 2) Tell the computer what to do if the condition is true or false.
  - ⇨ A decision always has a *Boolean value* or true/false answer.

The syntax for a decision uses the ***if*** statement.

Making Decisions
Performing Comparisons

Relational operators compare two items called operands.

- Syntax: *operand1 operator operand2*

Comparison operators in Java:

- > - Greater than
- >= - Greater than or equal
- < - Less than
- <= - Less than or equal
- == - Equal (Note: Not "=" which is used for assignment!)
- != - Not equal

The result of a comparison is a ***Boolean value*** which is either ***true*** or ***false***.

Making Decisions
Example Comparisons

```
int j=25, k = 45;
double d = 2.5, e=2.51;
boolean result;

result = (j == k);   // false
result = (j <= k);   // true
result = (d == e);   // false (rounding!)
result = (d != e);   // true
result = (k >= 25);  // true
result = (25 == j);  // true
result = (j > k);    // false
result = (e < d);    // false
j = k;
result = (j == k);   // true

// Note: Never compare doubles using "==" due to
//        precision and rounding problems.
```

Making Decisions
Comparing Strings and Objects

Comparing strings and objects is different than numbers.

- Operators such as <, > are not useful for strings and objects.
- Operator "==" is defined but it is not very useful.
  - The "==" operator compares if two string/object references refer to the same object **NOT** if the string/object has the same value.

Compare strings using **equals()** and **compareTo()** methods:

```
String str1 = "abc", str2="def";
str1.equals(str2);       // True if str1 is equal to str2
str1.equalsIgnoreCase(str2); // Comparison without case
str1.compareTo(str2); // will be < 0 if str1 < str2
str1.compareTo(str2); // will be > 0 if str1 > str2
str1.compareTo(str2); // will be = 0 if str1 equals str2
```

Making Decisions
Example String Comparisons

```
public class TestStringComparisons
{  public static void main(String[] args)
   {  String st1 = "Hello", st2="Hell", st3="Test",st4;

      System.out.println(st1.equals(st2));       // false
      System.out.println(st1.compareTo(st2));    // 1
      System.out.println(st2.compareTo(st1));    // -1
      System.out.println(st3.compareTo(st1));    // 12
      System.out.println(st3.compareTo("ABC")); // 19
      st4 = st1.substring(0,4);
      System.out.println(st2.equals(st4));       // true
      System.out.println(st2.compareTo(st4));    // 0
      st4 = st4.toUpperCase();
      st2 = st2.toLowerCase();
      System.out.println(st2.equals(st4));       // false
      System.out.println(st2.equalsIgnoreCase(st4));
                                                 //true
   }
}
```

String Comparisons

Question: What is the output of this code?

```
String str, str2;
Scanner sc = new Scanner(System.in);
str = sc.nextLine(); // User enters: abc
str2 = sc.nextLine();// User enters: abc

if (str == str2)
   System.out.print("equal");
else
   System.out.print("not equal");
```

A) `equal`

B) `not equal`

Making Decisions
If Statement

To make decisions with conditions, we use the ***if*** statement.

- If the condition is true, the statement(s) after **`if`** are executed otherwise they are skipped.
- If there is an **`else`** clause, statements after **`else`** are executed if the condition is false.

Syntax:

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

OR

```
if (condition)
   statement;
else
   statement;
```

Example:

```
if (age > 19)
   teenager=false;
```

OR

```
if (age > 19)
   teenager=false;
else
   teenager=true;
```

Making Decisions
Block Syntax

Currently, using our if statement we are only allowed to execute one line of code (one statement).

- What happens if we want to have more than one statement?

We use the ***block syntax*** for denoting a multiple statement block. A block is started with a "**{**" and ended with a "**}**".

- All statements inside the brackets are grouped together.

Example:

```
if (age > 19)
{  teenager=false;
   hasLicense=true;
   ...
}
```

We will use block statements in many other situations as well.

Making Decisions
If Statement Example

```
int age;
boolean teenager, hasLicense=false;
System.out.print("Enter your age: ");
Scanner sc = new Scanner(System.in);
age = sc.nextInt();

if (age > 19)
{  teenager = false;
   hasLicense = true;
}
else if (age < 13)
{  teenager = false;
   hasLicense = false;
}
else
{  teenager = true;  // Do not know if have license
}
System.out.println("Is teenager: "+teenager);
System.out.println("Has license? "+hasLicense);
```

Making Decisions

Question: What is the output of this code?

```
int num=10;

if (num > 10)
   System.out.println("big");
else
   System.out.println("small");
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (2)

Question: What is the output of this code?

```
int num=9;

if (num != 10)
    System.out.print("big");
System.out.println("small");
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (3)

Question: What is the output of this code?

```
int num=10;

if (num == 10)
{   System.out.print("big");
    System.out.println("small");
}
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions
Nested If Statement

We ***nest*** `if` statements for more complicated decisions.

- Verify that you use blocks appropriately to group your code!

Example:

```
if (age > 16)
{  if (sex == "male")
   {  System.out.println("Watch out!");
   }
   else
   {  System.out.println("Great driver!");
   }
}
else
{  System.out.println("Sorry! Too young to drive.");
}
```

Making Decisions
Nested If Statement Example

```
public class NestedIf
{  public static void main(String[] args)
   {  double salary, tax;
      String married;
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter M=married, S=single: ");
      married=sc.next();
      System.out.print("Enter your salary: ");
      salary=sc.nextDouble();

      if (married.equals("S"))
      {  // Single person
         if (salary > 50000)
            tax = salary*0.5;
         else if (salary > 35000)
            tax = salary*0.45;
         else
            tax = salary*0.30;
      }  // End if single person
```

Making Decisions
Nested If Statement Example

```
        else if (married.equals("M"))
        {  // Married person
           if (salary > 50000)
              tax = salary*0.4;
           else if (salary > 35000)
              tax = salary*0.35;
           else
              tax = salary*0.20;
        }  // End if married person
        else  // Invalid input
           tax = -1;

        if (tax != -1)
        {  System.out.println("Salary: "+salary);
           System.out.println("Tax: "+tax);
        }
        else
           System.out.println("Invalid input!");
    }
}
```

Nested Conditions and Decisions Dangling Else Problem

The ***dangling else problem*** occurs when a programmer mistakes an else clause to belong to a different if statement than it really does.

- Remember, blocks (brackets) determine which statements are grouped together, not indentation!

Example:

Incorrect

```
if (country == "US"))
   if (state == "HI"))
      shipping = 10.00;
else  // Belongs to 2nd if!
   shipping = 20.00; // Wrong!
```

Correct

```
if (country == "US")
{  if (state == "HI")
      shipping = 10.00;
}
else
   shipping = 20.00;
```

Nested Conditions and Decisions
Boolean Expressions

A ***Boolean expression*** is a sequence of conditions combined using AND (**&&**), OR (**||**), and NOT (**!**).

- Allows you to test more complex conditions
- Group subexpressions using parentheses

Syntax: *(expr1)* **&&** *(expr2)* - expr1 AND expr2

(expr1) **||** *(expr2)* - expr1 OR expr2

!(expr1) - NOT expr1

Examples:

```
var b;

1) b = (x > 10) && !(x < 50);
2) b = (month == 1) || (month == 2) || (month == 3);
3) if (day == 28 && month == 2)
4) if !(num1 == 1 && num2 == 3)
5) b = ((10 > 5 || 5 > 10) && ((10>5 && 5>10));// False
```

Boolean Expressions

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = (x > 10) || (y < 20);
System.out.println(result);
```

A) `true`

B) `false`

Boolean Expressions (2)

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = !(x != 10) && (y == 20);
System.out.println(result);
```

A) `true`

B) `false`

Boolean Expressions (3)

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = (x >= y) || (y <= x);
System.out.println(result);
```

A) `true`

B) `false`

Making Decisions (4)

Question: What is the output of this code?

```
int num=12;

if (num >= 8)
    System.out.print("big");
   if (num == 10)
       System.out.print("ten");
else
    System.out.print("small");
```

A) `big`

B) `small`

C) `bigsmall`

D) `ten`

E) `bigten`

Making Decisions (5) Boolean Expressions

Question: What is the output of this code?

```
int x = 10, y = 20;

if (x >= 5)
{  System.out.print("bigx");
   if (y >= 10)
       System.out.print("bigy");
}
else if (x == 10 || y == 15)
   if (x < y && x != y)
       System.out.print("not equal");
```

A) bigx

B) bigy

C) bigxnot equal

D) bigxbigynot equal

E) bigxbigy

Making Decisions
Switch Statement

There may be cases where you want to compare a single integer value against many constant alternatives. Instead of using many if statements, you can use a ***switch*** statement.

- If there is no matching case, the default code is executed.
- Execution continues until the **break** statement. (Remember it!)
- Note: You can only use a switch statement if your cases are **integer numbers**. (Characters ('a', 'b',...,) are also numbers.)

Syntax:

```
switch (integer number)
{   case num1: statement break;
    case num2: statement break;
    ...
    default:   statement break;
}
```

Making Decisions
Switch Statement Example

```
public class TestSwitch
{  public static void main(String[] args)
   {  int num;
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter a day number: ");
      num=sc.nextInt();
      switch (num)
      {  case 1: System.out.println("Sunday"); break;
         case 2: System.out.println("Monday"); break;
         case 3: System.out.println("Tuesday"); break;
         case 4: System.out.println("Wednesday"); break;
         case 5: System.out.println("Thursday"); break;
         case 6: System.out.println("Friday"); break;
         case 7: System.out.println("Saturday"); break;
         default: { System.out.println("Invalid day!");
                    System.out.println("Valid #'s 1-7!");
                  } break;
      }
   }
}
```

Switch Statement

Question: What is the output of this code?

```
int num=2;

switch (num)
{  case 1: System.out.print("one"); break;
   case 2: System.out.print("two"); break;
   case 3: System.out.print("three"); break;
   default: System.out.print("other"); break;
}
```

A) one

B) two

C) three

D) other

Switch Statement (2)

Question: What is the output of this code?

```
int num=1;

switch (num)
{  case 1: System.out.print("one");
   case 2: System.out.print("two");
   case 3: System.out.print("three"); break;
   default: System.out.print("other");
}
```

A) one

B) onetwo

C) onetwothree

D) other

E) onetwothreeother

Decision Practice Questions

1) Write a program that reads an integer *N*.

- If *N* < 0, print "Negative number", if *N* = 0, print "Zero", If *N* > 0, print "Positive Number".

2) Write a program that reads in a number for 1 to 5 and prints the English word for the number. For example, 1 is "one".

3) Write a program to read in your name and age and print them. Your program should print "Not a teenager" if your age is greater than 19 or less than 13, otherwise print "Still a teenager".

Iteration and Looping Overview

A computer does simple operations extremely quickly.

If all programs consisted of simple statements and decisions as we have seen so far, then we would never be able to write enough code to use a computer effectively.

To make a computer do a set of statements multiple times we program ***looping structures***.

A ***loop*** repeats a set of statements multiple times until some condition is satisfied.

- Each time a loop is executed is called an ***iteration***.

The While Loop

The most basic looping structure is the ***while*** loop.

A while loop continually executes a set of statements **while** a condition is true.

Syntax:

```
while (<condition>)
{  <statements>
}
```

Example:

```
int j=0;
while (j <= 5)
{  j=j+1;
   System.out.println(j);
}
```

The ++ and -- Operators

It is very common to subtract 1 or add 1 from the current value of an integer variable.

There are two operators which abbreviate these operations:

- ++- add one to the current integer variable
- -- - subtract one from the current integer variable

Example:

```
int j=0;

j++;      // j = 1;  Equivalent to j = j + 1;
j--;      // j = 0;  Equivalent to j = j - 1;
```

The For Loop

The most common type of loop is the ***for loop***. Syntax:

```
for (<initialization>; <continuation>; <next iteration>)
{  <statement list>
}
```

Explanation:

- 1) initialization section - is executed once at the start of the loop
- 2) continuation section - is evaluated *before* every loop iteration to check for loop termination
- 3) next iteration section - is evaluated *after* every loop iteration to update the loop counter

Iteration & Looping
The For Loop

Although Java will allow almost any code in the three sections, there is a typical usage:

```
for (i = start; i < end; i++)
{   statement
}
```

Example:

```
int i;

for (i = 0; i < 5; i++)
{   System.out.println(i);      // Prints 0 to 4
}
```

Java Rules for Loops

The iteration variable is a normal variable that must be declared, but it has the special role of controlling the iteration.

- i, j, and k are the most common choices due to convention and because they are short.

The starting point of the iteration can begin anywhere, including negative numbers.

The continuation/termination test must be an expression that results in a Boolean value. It should involve the iteration variable to avoid an ***infinite loop***.

The next iteration can have any statements, although usually only use the step size to change iteration variable.

- The step size can be positive or negative and does not always have to be 1.

Common Problems – Infinite Loops

Infinite loops are caused by an incorrect loop condition or not updating values within the loop so that the loop condition will eventually be false.

Examples:

```
int i;

for (i=0; i < 10; i--)      // Should have been i++
{  System.out.println(i);   // Infinite loop: 0,-1,-2,..
}

i = 0;
while (i < 10)
{  System.out.println(i);   // Infinite loop: 0,0,0,..
}                           // Forgot to change i in loop
```

Common Problems – Using Brackets

A one statement loop does not need brackets, but we will ***always use brackets***. Otherwise problems may occur:

```
int i=0;
while (i <= 10)
   System.out.println(i);  // Prints 0 (infinite loop)
   i++;                    // Does not get here…
// Forgot brackets { and } - i++ not in loop!
```

Do not put a semi-colon at the end of the loop:

```
int i;

for (i=0; i <= 10; i++);   // Causes empty loop
{  System.out.println(i);  // Prints 11
}
```

Common Problems – Off-by-one Error

The most common error is to be "***off-by-one***". This occurs when you stop the loop one iteration too early or too late.

Example:

- This loop was supposed to print 0 to 10, but it does not.

```
for (i=0; i < 10; i++)
   document.write(i);  // Prints 0..9 not 0..10
```

Question: How can we fix this code to print 0 to 10?

Common Problems – Iteration Variable

Scope Issues: It is possible to declare a variable in a for loop but that variable goes out of scope (disappears) after the loop is completed.

```
int i;
for (i=0; i <= 10; i++)
{  System.out.println(i);  // Prints 0..10
   ...
}
System.out.println(i);     // Prints 11
```

Other approach:

```
for (int i=0; i <= 10; i++)// Declare i in for loop
{  System.out.println(i);  // Prints 0..10
   ...
}
System.out.println(i);     // Not allowed - i does
                           // not exist outside loop
```

For Loops

Question: What is the output of this code?

```
int i;

for (i=0; i <= 10; i++);
   System.out.print(i);
```

A) nothing

B) error

C) `11`

D) The numbers 0, 1, 2, …, 10

For Loops

Question: What is the output of this code?

```
int i;

for (i=0; i < 10; i++)
   System.out.print(i);
```

A) nothing

B) error

C) The numbers 0, 1, 2, …, 9

D) The numbers 0, 1, 2, …, 10

For Loops

Question: What is the output of this code?

```
int i;

for (i=2; i < 10; i--)
   System.out.print(i);
```

A) nothing

B) infinite loop

C) The numbers 2, 3, 4, …, 9

D) The numbers 2, 3, 4, …, 10

The do..while Loop

The last looping structure called a ***do..while*** loop. The do..while loop is similar to the while loop except that the loop condition is tested at the bottom of the loop instead of the top.

- This structure is useful when you know a loop must be executed at least once, but you do not know how many times.

Syntax:

```
do
{  statement
} while (condition);
```

Example:

```
do
{  num = num / 2;
} while (num >= 0);
```

Loop Nesting

Similar to decisions statements such as if and switch, it is possible to nest for, while, and do..while loops.

- Note that the loops do not all have to be of the same type.
  - i.e. You can have a for loop as an outer loop, and a while loop as an inner loop.

Be very careful to include correct brackets when nesting loops.

- It is a good idea to always include brackets in your code to make your code more readable and prevent mistakes.

Nested For/While Loop Example

```
// Prints N x N matrix until N = -1

public class NestedForWhile
{  public static void main(String[] args)
   {  int i, j, num;
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter a matrix size: ");
      num=sc.nextInt();
      while (num != -1)
      {
         for (i=1; i <= num; i++)
         {  for (j=1; j <= num; j++)
               System.out.print(j+" ");   // No brackets!
            System.out.println();
         }
         System.out.print("Enter a matrix size: ");
         num=console.readInt();
      }
   }
}
```

Advanced Topic: Break Statement

What happens if you want to exit a loop before the end?

- You can use the ***break*** statement to immediately exit the current loop block.
  - Note: The break statement exits the current loop. If you have a nested loop, you will need multiple break statements to get out of all loops.

Example:

```
while (true)
{   System.out.print("Enter a matrix size: ");
    num=console.readInt();
    if (num == -1)
       break;
    ...
}
// After break - execution starts here
```

Advanced Topic: Continue Statement

What happens if you want to quickly skip back to the start of the loop (end the current iteration) while in the middle of the loop statements?

- You can use the ***continue*** statement to immediately stop the current loop iteration and start the next one.
  - Note: This is rarely used.

Example:

```
for (i=0; i < 5; i++)   // After continue, start i=3
{  if (i  == 2)
      continue;   // For some reason we don't like 2!
   System.out.println(i);
}

// Question: What is the better way to do this?
```

Looping Review

A loop structure makes the computer repeat a set of statements multiple times.

- for loop is used when you know exactly how many iterations to perform
- while loop is used when you keep repeating the loop until a condition is no longer true
- a do..while loop is used when a loop has to be performed at least once

When constructing your loop structure make sure that:

- you have the correct brackets to group your statements
- you do not add additional semi-colons that are unneeded
- make sure your loop terminates (no infinite loop)

Remember the operators ++ and -- as short-hand notation.

Continue Statement

Question: How many numbers are printed?

```
for (int i=2; i < 10; i++)
{  if (i % 2 == 0)
      continue;
   System.out.print(i);
}
```

A) 0

B) 4

C) 5

D) 9

Break Statement

Question: How many numbers are printed?

```
for (int i=2; i < 10; i++)
{  if (i > 4)
      break;
   System.out.print(i);
}
```

A) 9

B) 5

C) 4

D) 3

Practice Questions: Iteration

1) How many times does each loop execute:

a)
```
for(j=0; j <= 10; j--)
```
b)
```
for(j=0; j <= 10; j++)
```
c)
```
for(j=0; j < 10; j++)
```
d)
```
for(j=-10; j <= 10; j++)
```
e)
```
for(j=0; j <= 20; j=j+2)
```

2) Write a program to print the numbers from 1 to N.

- a) Modify your program to only print the even numbers.

3) Write a method that builds and prints an integer matrix of the form: (where N is given).

```
1 1 1 … 1
2 2 2 … 2
…
N N N … N
```

Conclusion

A ***decision*** is performed by evaluating a Boolean condition with an `if/else` statement.

A ***loop*** allows the repetition of a set of statements multiple times until some condition is satisfied.

- We will primarily use `for` loops that have 3 components:
  - initialization - setup iteration variable start point
  - continuation - use iteration variable to check if should stop
  - next iteration - increment/decrement iteration variable

Decision and loops can be nested.

Objectives

Java skills:

- Make decisions using if/else statement.
- Use Boolean variables to represent true/false.
- Use relational operators in conditions.
- Comparing Strings and Objects using equals and compareTo.
- Build complex conditions using AND, OR, and NOT.
- Switch statement
- Iteration using three loop constructs:
  - while statement
  - for statement
  - do…while statement
- Break and continue statements
- Nesting of if/else and iteration statements

Detailed Objectives

- Write decisions using the if/else statement.
- Define: Boolean, condition
- List and use the comparison operators.
- Explain the dangling else problem.
- Construct and evaluate Boolean expressions using AND, OR, and NOT.
- Explain why cannot use == with Strings/Objects.
- Define: loop, iteration
- Explain the difference between the while and for loops.
- Explain what ++ and -- operators do.
- Be able to use a for loop structure to solve problems.
- Be aware and avoid common loop problems.
- Define: infinite loop