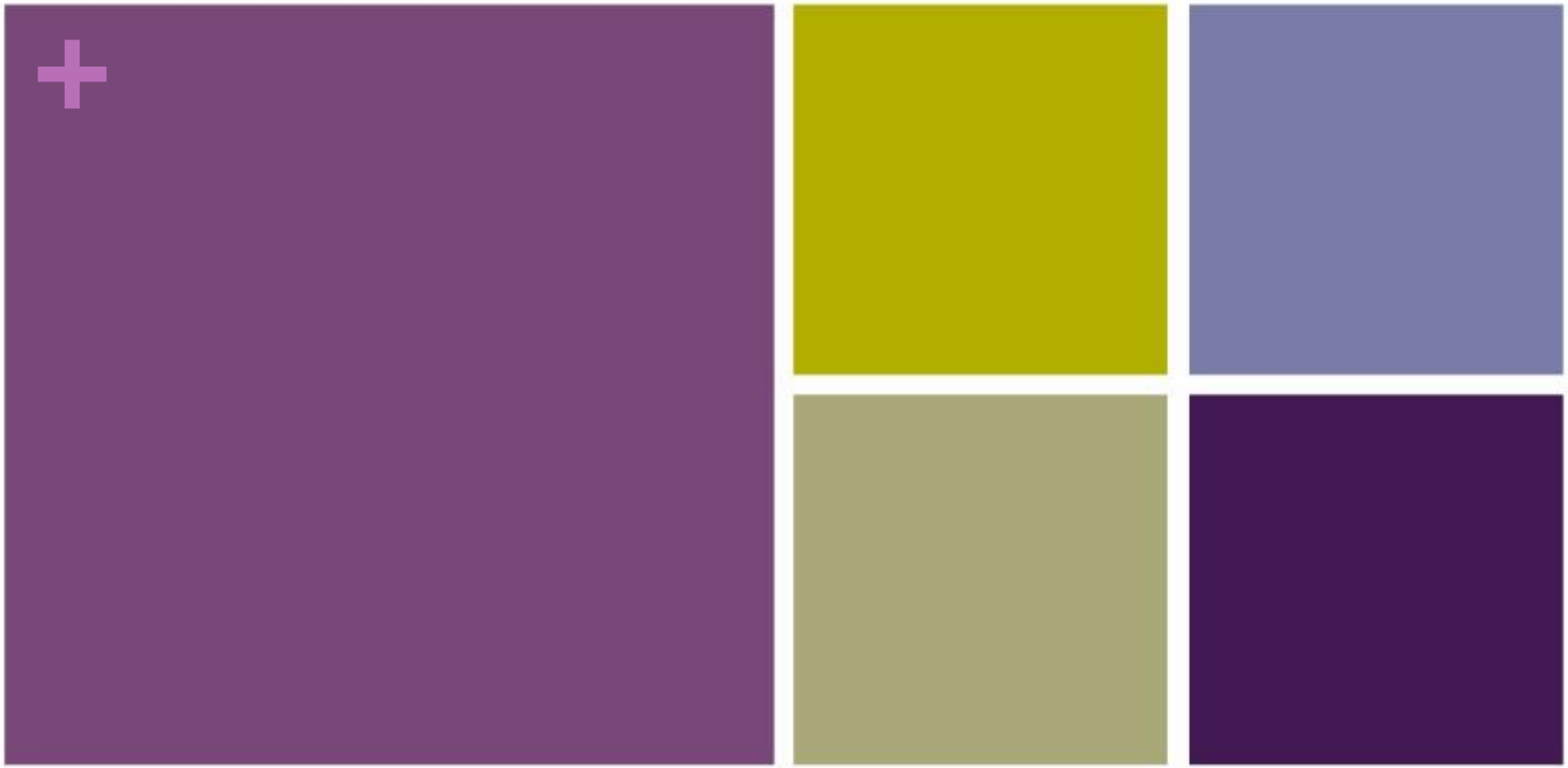


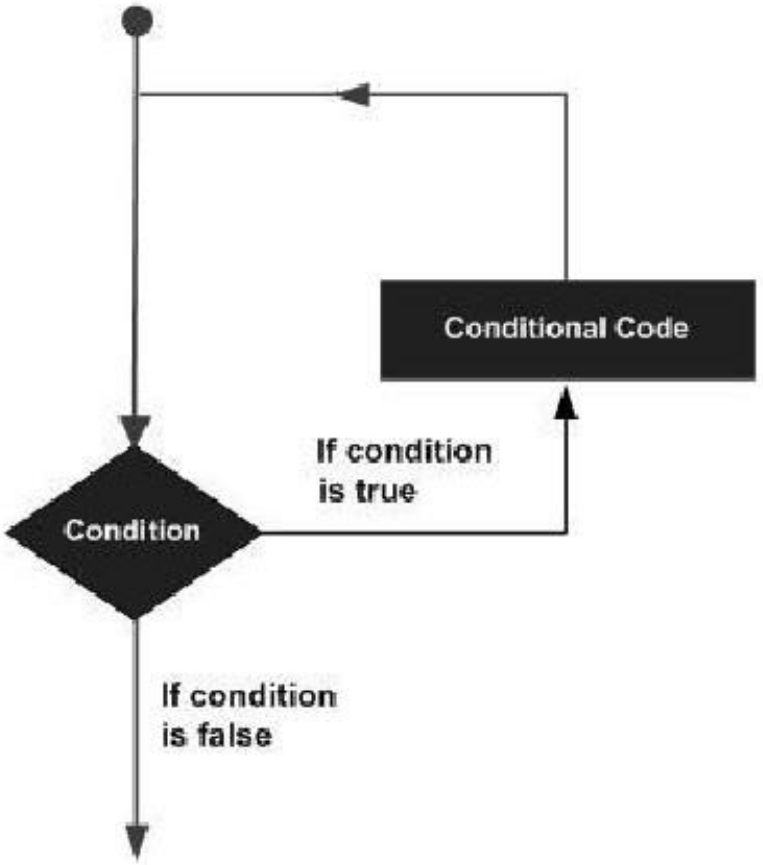
Repetition Structures (Loops)

Introduction to Computer Science CSCI-UA.0101

Lecture 5

Agenda - Day 5

- Brief Review
- Repetition Structures (loops)
- “while” loops
- Loop Design Strategies
- "break" and "continue"
- Sentinels
- The "do-while" loop
- The "for" loop
- Nested Loops



+

Repetition Structures

Repetition Structures

- A repetition structure is a programmatic construct that allows you to repeatedly execute a series of statements. For example, the statements:

```
System.out.println("hi");
System.out.println("hi");
System.out.println("hi");
```

- Can be re-written using a repetition structure:

```
int count = 0;
while (count < 3)
{
   System.out.println("hi");
   count++;
}
```

- We informally refer to repetition structures as "loops"

Types of Repetition Loops

- Java supports 3 different loop structures
  - The "for" loop
  - The "while" loop
  - The "do-while" loop
- You typically use the "for" loop when you know how many times the loop is going to repeat. (Examples?)
- You typically use the "while" loop when you don't know how many times the loop is going to repeat. (Examples?)
- Everything that you can do with a "for" can be done with a "while" loop; however, the opposite is not true. The "while" loop is harder to use.
- The "while" loop is similar to the "do-while" loop with one difference: the "while" is a pre-test loop; the "do-while" is a post-test loop. As a consequence the statements inside the "do-while" are executed at least once.

The “for” Loop

The "for" loop

- The "for" loop is an "all in one" loop that combines three common loop techniques into a single statement:
  - Initialization of your condition

```
// int counter = 0;
```

  - Loop continuation condition

```
// counter < 10
```

  - After iteration action

```
// counter++
```

The "for" loop

- Syntax:

```
for (int counter = 0; counter < 10; counter++)
{
     // statement(s)
}
```

- Note that in the above example we declared and defined 'counter' inside the loop. We could have just as easily defined 'counter' outside the loop, like this (just don't do both!):

```
int counter = 0;
for (counter = 0; counter < 10; counter++)
{
     // statement(s)
}
```

The "for" loop

- "for" loops can be used for any iteration task, but they excel at giving you the ability to iterate over a known range of values. For example, if you want to do something a million times you could easily set that up with using following "for" loop:

```
for (int i = 0; i < 1000000; i++)
{
    // statements
}
```

Accumulator Variables

- Many programming tasks require you to calculate the total of a series of numbers or the number of times you iterate through a loop. We can accomplish this by creating an accumulator variable that can be updated over the lifetime of the loop.
- Initialize your accumulator variables outside of your loops.
- Decide on a value you want to start your accumulator values at (0 works well if you are counting something)
- Use a self-referential assignment statement or augmented assignment statement when incrementing an accumulator variable. Example:
  - counter = counter + 1
  - counter += 1
  - counter++

Programming Challenge

- Write a program that asks a teacher how many grades a s/he wants to enter in the system. The program will then prompt the teacher to enter all the grades and then it will calculate the grade average for the class.

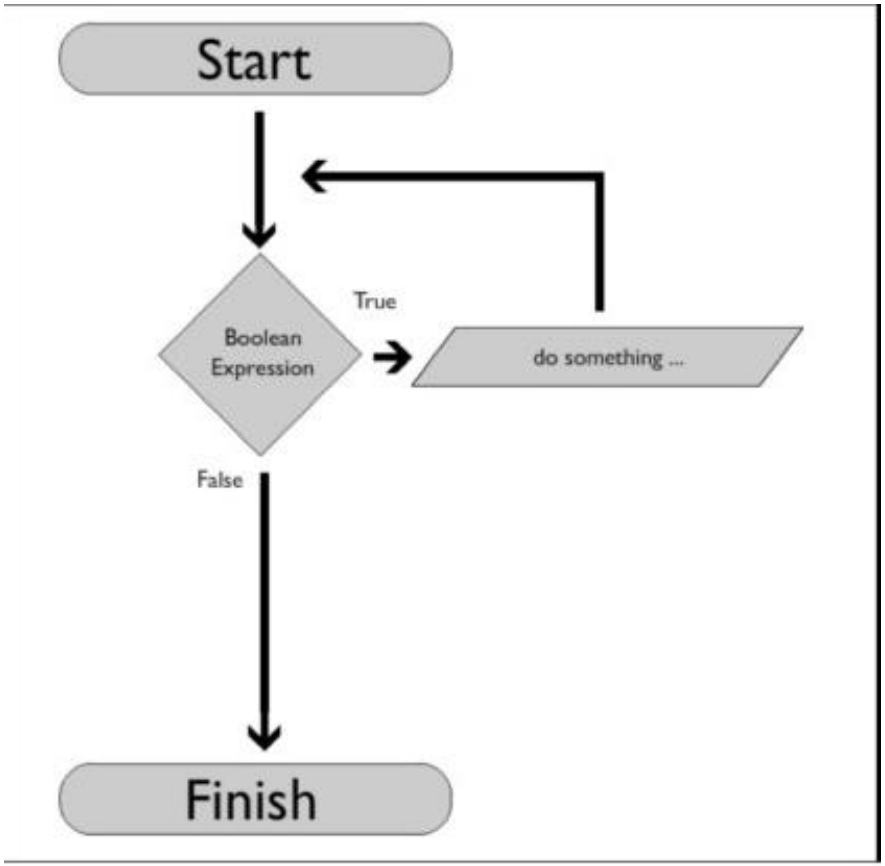
- **GradeAverageFor.java**

The “while” Loop

The "while" Loop

- The "while" loop is a construct that executes statements repeatedly as long as a condition evaluates to true
- Think of a while loop like an "if" statement, but with repetition
  - Evaluate a condition
  - If it evaluates to false, skip the attached block and continue with the program
  - If it evaluates to true:
    - Execute the attached block of statements
    - At the end of the block re-evaluate the condition. If it still evaluates to true, repeat the block a second time. If it evaluates to false skip the block and continue with the program.

The "while" Loop



The "while" Loop

- We refer to the process of going through a loop as an "iteration"
- If a loop cycles through 5 times then we say we have "iterated" through it 5 times
- The "while" loop is considered a "pre-test" loop, meaning that it only iterates upon the successful evaluation of its loop continuation condition
- This means that you always need to "set up" your loop prior to Java being able to work with it (i.e. setting up a control variable)

"while" Loop Syntax

```
while (condition)
{
  // statement(s)
}
```

Example: How many times the loop iterates?

```
double num = 10.0;
while (num > 0.0)
{
      System.out.println(num);
}
```

Infinite Loops

- When working with a "while" loop there is nothing to prevent you from writing a Boolean condition that will never evaluate to false.We call this an "infinite loop" since it never stops executing.
- If this happens your loop will continue executing forever, or until you send an "interrupt" to Eclipse using the "stop" button in the console panel.
- With the exception of a few special cases you want to try and avoid writing infinite loops

Accumulator Variables

- Many programming tasks require you to calculate the total of a series of numbers or the number of times you iterate through a loop. We can accomplish this by creating an accumulator variable that can be updated over the lifetime of the loop.
- Initialize your accumulator variables outside of your loops.
- Decide on a value you want to start your accumulator values at (0 works well if you are counting something)
- Use a self-referential assignment statement or augmented assignment statement when incrementing an accumulator variable. Example:
  - counter = counter + 1
  - counter += 1
  - counter++

Sentinels

- A sentinel is a pre-defined value that can be used to indicate that a loop should cease iterating. Example:

```
// create a scanner
Scanner input = new Scanner(System.in);

// create a sum accumulator variable
int sum = 0;

// continually accept values from standard input as long as the number
// provided is not zero
while (true)
{
        // get a value
        System.out.print("Enter an integer (0 ends): ");
        int value = input.nextInt();

        // is this our sentinel (0)?
        if (value == 0)
        {
                break;
        }

        // add to our sum
        sum += value;
        }

// report sum to user
System.out.println("Your total is: " + sum);
```

Programming Challenge

- Write a program that allows a teacher the enter grades until a sentinel value is entered. The program will then show the number of grades entered and will calculate the grade average for the class.

- **GradeAverageWhile.java**

Programming Challenge

- Write a number guessing game that repeatedly asks the user to guess an integer between 1 and 100
- If the user guesses the number, end the loop
- Otherwise report to them whether they are too low or too high
- Keep track of the # of attempts it took the user before they successfully guessed the number
- **GuessingGame.java**

The "off by one" error

- It's easy to execute a loop one more time than necessary. For example:

```
// this will iterate 101 times
int counter = 0;
while (counter <= 100)
{
     // statements
     counter++;
}
```

Loop Design Strategies

Loop Design Strategies

- Identify statements that need to be executed
- Wrap the statements inside a loop structure

```
while (true)
{
	// statements
}
```

- Code the loop continuation condition and add in appropriate statements for controlling the loop. You will most likely also need to set up control structures outside the loop.

```
// set up control structure

while (loop continuation condition)
{
	// statements

	// statements to control the loop
}
```

+

“break” and “continue”

The "break" Command

- The break command can be used to immediately end a loop
- It's not required to ever use break - you can always simulate its function by adjusting your loop control condition
- Example:

```
int sum = 0;
while (true)
{
	sum += (int) (Math.random() * 11);
	System.out.println("sum = " + sum);
	if (sum % 13 == 0)
	{
		System.out.println("Found a multiple of 13!");
		break;
	}
	System.out.println("Looping	");
}
```

The "continue" Command

- The "continue" command can be used to cause a loop to end its current iteration, but not end the loop. For example:

```
// sum up all numbers between 1 and 100, except multiples of 10

int counter = 0;
int sum = 0;
while (counter < 100)
{
        // increase counter
        counter++;

        // is this a multiple of 100?
        if (counter % 10 == 0)
        {
                // skip it
                continue;
        }
        else
        {
                sum += counter;
        }
}

System.out.println("Sum of 1 - 99: " + sum);
```

The “do-while” Loop

The "do-while" loop

- The "do-while" loop is almost the same as a "while" loop except that it will execute its block before it evaluates its condition. For example:

```
do
{
      // statements
}
while (condition);
```

- "do-while" loops are useful when you know that you want your loop to execute at least one time.

- Note: the "do-while" loop requires a semi-colon after the condition at the end of the loop!

Programming Challenge

- Write an input validation loop that asks the user to provide an integer between 1 and 100

- **InputValidation.java**

Nested Loops

Nested Loops

- A nested loop is a "loop inside another loops"
- We usually refer to nested loops in terms of their "outer" and "inner" loops. The outer loop is the first loop that is encountered in your program, and the inner loop is the loop that is nested inside of the outer loop. For example:

```
for (int outer = 0; outer < 10; outer++)
{
	// statement(s)

	for (int inner = 0; inner < 10; inner++)
	{
		// statement(s)
	}

	// statement(s)
}
```

Nested Loops

- In a nested loop configuration the inner loop will iterate through its full range one time for each iteration of the outer loop. For example:

```
for (int outer = 0; outer < 3; outer++)
{
      for (int inner = 0; inner < 3; inner++)
      {
            System.out.println(outer + "-" + inner);
      }
}
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

Programming Errors

Types of Errors

- **Syntax errors**: The code does not follow the rules of the language; for example, a single quote is used where a double quote is needed; a semi-colon is missing; a keyword is used as a variable name.

- **Runtime errors**: In this case, your code is fine but the program does not run as expected (it "crashes"). For example, if your program is meant to divide two numbers, but does not test for a zero divisor, a run-time error would occur when the program attempts to divide by zero.

- **Logic errors**: These can be the hardest to find. In this case, the program is correct from a syntax perspective; and it runs; but the result is unanticipated or outright wrong. For example, if your program prints "2+2 = 5" the answer is clearly wrong

Example Errors

```
System.out.printtln(“hi”);
```

Example Errors

```
System.out.printtln("hi");
```

Syntax error:

"printtln" should be "println"

Example Errors

```
public class Welcome {
  public static main (String[] args) {
     System.out.println(“hi there”)
     System.out.println(“bye!”)
  }

}
```

Example Errors

```
public class Welcome {
  public static main (String[] args) {
     System.out.println(“hi there”)
     System.out.println(“bye!”)
  }

}
```

Syntax Errors:

1. The “void” keyword is missing before the main method declaration
2. Missing semicolons after the two System.out.println method calls

Example Errors

```
import java.util.Scanner;

public class Welcome {
  public static void main (String[] args) {
     Scanner numberScanner = new Scanner(System.in);
     int a = numberScanner.nextInt(); // a = 0
     System.out.println( 100 / a );
  }

}
```

Example Errors

```
import java.util.Scanner;

public class Welcome {
  public static void main (String[] args) {
     Scanner numberScanner = new Scanner(System.in);
     int a = numberScanner.nextInt(); // a = 0
     System.out.println( 100 / a );
  }

}
```

Runtime Error:

Cannot divide a number by zero.

Example Errors

```
public class Welcome {
  public static main (String[] args) {
     System.out.print(“Test score average: “);
     System.out.println( (100+95+87) / 2 );
  }

}
```

Example Errors

```
public class Welcome {
  public static void main (String[] args) {
     System.out.print("Test score average: ");
     System.out.println( (100+95+87) / 2 );
  }

}
```

Logic Error:

Program will run, but the answer that is being given is incorrect