Iteration Structures

Basic Elements
- Initialization
- Loop Test
- Main Work
- Update
- Finalization (maybe)

Counting loop flowchart
- set count to 0
- if count <= 10
  - display count
- increment count
- end if

the while loop
```java
int count = 0;
while ( count <= 10 ) {
    System.out.println( count );
    count++;
}
```
What will be displayed?

Input validation loop flowchart
- get number from user
- if number invalid
  - get another number
- else
  - true
- end if

Example: input valid grade
```java
import uwcse.io;
userInput = new Input( ); //from uwcse.io
double grade = userInput.readDouble("Enter grade: ");
while ( grade < 0 || grade > 4.0 ) {
    grade = userInput.readDouble("Error: re-enter grade");
}
//now can use grade in the rest of the code
```
Will keep repeating while the input is incorrect.
Types of Loops

Java has 3 structures for looping:
- `while` loop
- `do-while` loop
- `for` loop

Any loop can be rewritten as any other, but choosing the best loop for a task will make it easy to understand.

What makes these loops different?

- `while` loop – pretest loop (executes 0 or more times)
- `for` loop – pretest loop; best for counting
- `do-while` loop – posttest loop (executes 1 or more times)

Do-while loop

Suppose you wanted to use a game program. After each play, it asks if you want to play again.

```java
boolean anotherGame;
do{
    playGame();
anotherGame=getInput();
}while (anotherGame);
```

Counting

The `for` loop is best for counting

Looking back on a previous example, you can see three critical numbers:
- initial value of the count (0)
- increment value of the count (1)
- final value of the count (10)

```java
int count = 0;
while (count <= 10) {
    System.out.println(count);
count = count + 1;
}
```

For loop

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

What’s the order of execution?

Loop Variables

Java supports smallest scope, a variable that can ONLY be used inside a loop

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

```
int count = 0;
while (count <= 10) {
    System.out.println(count);
count = count + 1;
}
```

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

```
int count;
for( count = 0; count <= 10; count++ ) {
    System.out.println(count);
    System.out.println(i + z);
}
```
Choosing a loop structure

- If you know in advance how many iterations will be required (known as a **definite loop**), consider using a **for loop**.
- When the decision to continue repetition is based on a condition or a flag, so that the number of repetitions is not known (an **indefinite loop**), consider using a **while loop**.
  - But, if the body of the loop must always execute at least once, consider using a **do-while loop**.

Nested Control Structures

- You can nest any control structure with in any other
  - Loops within loops
  - if statements within if statements
  - Loops within if statements
  - if statements within loops
- The inner structure is considered 1 statement.

Nested Loops

Let's use nested loops to create a pattern, where only one '*' may be printed at a time – use only
  - System.out.print(" ** ");
  - System.out.println(");

```
****
****
****
```

Nested Loops (2)

Now try this pattern:

```
*
**
***
****
*****
```

Common loop errors

```java
for(int i = 1; i < 10; i++);
System.out.println(" Hello ");
```

```java
int i = 0;
while( i < 10 )
System.out.println( i );
i++;
```

```java
for(int i=1; i!=10; i+=2)
System.out.println(" ** ");
```

Don't count with doubles

```java
for( double x = 0; x <= 1; x += 0.1 )
System.out.println( x );
```

Actual output:

```
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
```

What's the problem?

Just like z cannot be stored exactly in the statement:
```
double z = 1.0 / 3.0;
```
many decimal values cannot be stored exactly, so you see truncation!