CSC 142

Iterations

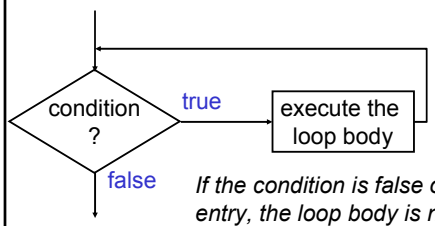
[Reading: chapter 10]

Iterations in Java

- Java provides three constructs to iterate
  - while loop
  - do-while loop
  - for loop
- Any loop written using one construct can always be rewritten using another one. Of course, it might not improve the clarity of your program!
- Which one you choose depends on what you think is clearer for the problem at hand.

while loop

```
while (condition)
{ // loop body
  statement; // 1 or more statements
}
```



If the condition is false on loop entry, the loop body is never executed (not even once).

while loop example

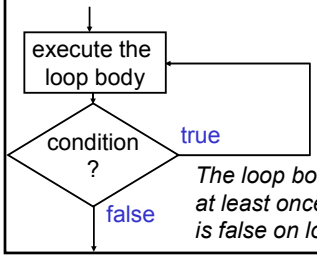
- Sum all of the integers from 1 to 100

```
int i=1;
int sum=0;
while(i<=100)
{
  sum += i;
  i++;
}
System.out.println("sum="+sum);
```


do-while loop

```
do
{ // loop body
  statement; // 1 or more statements
}
while (condition);
```

don't forget



The loop body is always executed at least once, even if the condition is false on loop entry

do-while loop example

- Let a user play a game as many times as she wants

```
boolean play;
do
{
  playGame();
  // Ask if the user wants to play again
  // details are in method anotherGame
  play = anotherGame();
}while(play);
```

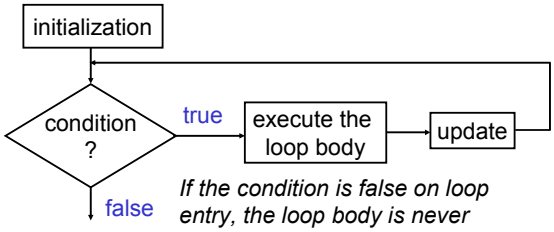
Sentinel loop

- When we don't know how many inputs there are

```
//Compute the average of a set of grades
int count=0; double sum=0.0, grade;
Input input = new Input();
do{
 // a grade is between 0.0 and 4.0
 System.out.print("Grade(-1 to stop): ");
 grade=input.readDouble();
 if (grade!=-1) {
     sum+=grade;
     count++;}
}while(grade!=-1); //-1 sentinel value
if (count>0)
 System.out.print("Average="+sum/count);
```


for loop

```
for(initialization;condition;update)
{ // loop body
  statement; // 1 or more statements
}
```



If the condition is false on loop entry, the loop body is never executed (not even once).

for loop example (1)

- Display the following pattern

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

```
lines=3; cols=5;
for(line=1; line<=lines; line++)
{
 // if 1 statement only,{ and } are optional
 // print cols "*"
 for(col=1; col<=cols; col++)
     System.out.print("*");
 // next line
 System.out.println();
}
```


for loop example (2)

- Display the following pattern

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

```
lines=3;
for(line=1; line<=lines; line++)
{
 // print line "*"
 for(col=1; col<=line; col++)
     System.out.print("*");
 // next line
 System.out.println();
}
```


for loop example (3)

- Display the following pattern

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

```
lines=3;
for(line=1; line<=lines; line++)
{
 //print lines-line spaces
 for (col=1; col<=lines-line; col++)
     System.out.print(" ");
// print line "*"
for(col=1; col<=line; col++)
     System.out.print("*");
 // next line
 System.out.println();
}
```


Which loop construct should I use?

- for loop: when you know number of iterations before entering the loop
  - initialization and update are clearly visible
- while loop: when the number of iterations depends on what happens inside of the loop body
- do-while: when the loop body is executed at least once

Loop variable

- Declare the loop variable in the initialization statement of the for loop, e.g.

```
for(int i=1; i<=10; i++)
    System.out.print(i+" squared is "+i*i);
```

- Warning: i can only be used within the loop body. i doesn't exist outside the loop body.
- Drop the braces { } if there is only 1 statement in the loop body

Some loop pitfalls

```
int i=0, int sum=0;
for(i=0; i<=10; i++);
  sum+=i;
```

The loop ends here. sum+=i; is executed only once. sum is 11.

```
int i=0, int sum=0;
while(i<=10);
{
  sum+=i;
  i++;
}
```

The loop ends here. i is never updated. The loop runs forever

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

i is never 10. The loop runs forever

Don't count with doubles

```
DecimalFormat d;
d = new DecimalFormat("0.000000000000000");
for(double x=0; x<10; x+=0.2)
  System.out.println("x="+d.format(x));
```

- What is printed?

```
x=0.000000000000000
...
x=9.599999999999998
x=9.799999999999997
x=9.999999999999996
```

x should be 10 and not printed. Get an extra iteration

Parsing a String

- Display each word in a String
- Use a StringTokenizer object (in java.util)

```
// Define the String
String s = "Hello, how are you?";
StringTokenizer st = new StringTokenizer(s);
// By default, st uses a whitespace delimeter
// for parsing (other options are available)
while(st.hasMoreTokens())
 System.out.println(st.nextToken());
// The words "Hello,", "how", "are", "you?"
// are printed (one on each line).
```


Iterating through a list

- Input a list of integers, sort and print the list

```
// Ordering a list of positive integers
ArrayList a = new ArrayList();//an implementation of
int i;                        //a list (in java.util)
Input input = new Input();
// Get the integers
do{
 i=input.readInt("Enter an integer(<0 to stop)");
 if (i>=0) a.add(new Integer(i)); //use a wrapper
}while(i>=0);
// Order the collection
Collections.sort(a); //in java.util
// Iterate through the collection and print it
Iterator it = a.iterator(); //in java.util
while(it.hasNext())
 System.out.print(it.next()+" ");
```


break statement

- In a loop context, exit the innermost loop that contains the break statement

```
for(int i=1; i<10; i++)
{
 for(int j=1; j<10; j++)
  {
     for(int k=1; k<10; k++)
     {
         // some code…
         if (k>5) break; // exit the k loop
     }
    // The break statement transfers
    // execution here
  }
}
```

labeled break statement

- In a loop context, exit the labeled loop that contains the break statement

```
for(int i=1; i<10; i++)
{
  myLabel: // label name
   for(int j=1; j<10; j++)
   {
      for(int k=1; k<10; k++)
      {
          // some code…
          // exit the loop labeled myLabel
         if (k>5) break myLabel;
      }
   }
   // execution resumes here
   // after the break
}
```


continue statement

- In a loop context, proceed with the next iteration of the innermost loop that contains the continue statement

```
for(int i=1; i<5; i++)
{
  System.out.println("Begin: i="+i);
  if (i%2==0) continue; //If i is even, don't
                        //finish the execution
                        //of the loop.
                        //Proceed with the
                        //loop update.
 System.out.println("End: i="+i);
}
```

What is printed?

labeled continue statement

- In a loop context, proceed with the next iteration of the labeled loop that contains the continue statement

```
myLabel: // label name
  for(int i=1; i<4; i++)
  {
    for(int j=1; j<4; j++)
    {
       System.out.println("i="+i+" j="+j);
       if (j%2==0) continue myLabel;
    }
  }
```

- What is printed?

Warning

- Use continue and break carefully.
- The control flow is not easy to follow whenever a programmer uses lots of break and continue
- However, these statements provide an elegant way to get out of a set of nested loops.