FCDS

Programming I

Lecture 6: Loops (Part II)

Today's Lecture

- Fencepost Loop
- Nested `for` loops
- while and do-while loops
- Sentinel (indefinite) loops

A deceptive problem...

- Write code that prints each number from 1 to 5, separated by commas.

  should print:

```
 1, 2, 3, 4, 5
```

Flawed solutions

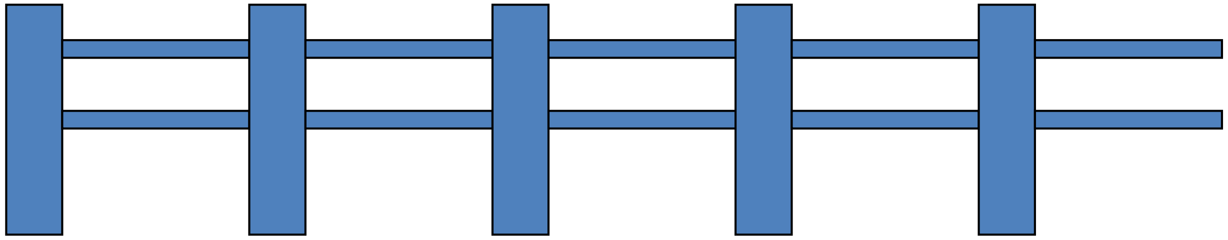
-
  ```
  for (int i = 1; i <= 5; i++) {
      System.out.print(i + ", ");
  }
  System.out.println();   // to end the line of output
  ```
  - Output:
    ```
    1, 2, 3, 4, 5,
    ```

-
  ```
  for (int i = 1; i <= 5; i++) {
      System.out.print(", " + i);
  }
  System.out.println();   // to end the line of output
  ```
  - Output:
    ```
    , 1, 2, 3, 4, 5
    ```

Fence post analogy

- We print *n* numbers but need only *n* - 1 commas.
- Similar to building a fence with wires separated by posts:
  - If we use a flawed algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

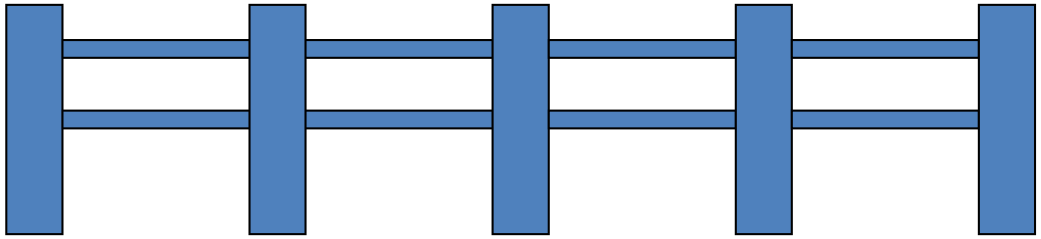
```
for (length of fence) {
    place a post.
    place some wire.
}
```



Fencepost loop

- Add a statement outside the loop to place the initial "post."
  - Also called a *fencepost loop* or a "loop-and-a-half" solution.

```
place a post.
for (length of fence - 1) {
    place some wire.
    place a post.
}
```



Fencepost method solution

```
System.out.print(1);
for (int i = 2; i <= max; i++) {
    System.out.print(", " + i);
}
System.out.println();       // to end the line
```

- Alternate solution: Either first or last "post" can be taken out:

```
for (int i = 1; i <= max - 1; i++) {
    System.out.print(i + ", ");
}
System.out.println(max);  // to end the line
```

Nested `for` loops

Nested loops

- **nested loop**: A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();    // to end the line
}
```

- Output:

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

Outer loop	inner loop
i=1	j goes from 1 to 10
i=2	j goes from 1 to 10
i=3	j goes from 1 to 10
i=4	j goes from 1 to 10
i=5	j goes from 1 to 10

i=6 <=5? No then done

- The outer loop repeats 5 times; the inner one 10 times.

Nested `for` loop exercise

- What is the output of the following nested `for` loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Output:

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

Outer loop	**inner loop**
`i=1`	`j goes from 1 to 1`
`i=2`	`j goes from 1 to 2`
`i=3`	`j goes from 1 to 3`
`i=4`	`j goes from 1 to 4`
`i=5`	`j goes from 1 to 5`

`i=6 <=5? No then done`

Common errors

- Both of the following sets of code produce *infinite loops*:

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}


for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Loop (Iteration) Statements

- ***Loop statements*** allow repeatedly executing a statement or a sequence of statements one or more times as long as some condition remains true.

- There are three loop statements in Java
  - ***for*** loop statement ✓
  - ***while*** loop statement ⇦
  - ***do-while*** loop statement ⇦

`while` loops

Categories of loops

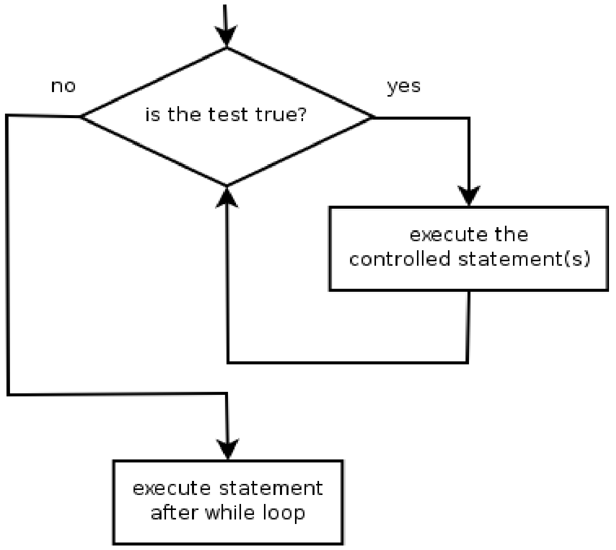
- **Count-controlled (definite) loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

- **Sentinel-controlled (indefinite) loop**: One where the number of times its body repeats is not known in advance.
    - Prompt the user until they type a negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user types "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

```
while (test) {
    statement(s);
}
```



no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

- Example:

```
int num = 1;                          // initialization
while (num <= 200) {                  // test
    System.out.print(num + " ");
    num = num * 2;                    // update
}
// output:  1 2 4 8 16 32 64 128
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
     System.out.println(count);


     count -- ;
 }
System.out.print( " Done "  ) ;
```

count

OUTPUT

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);

      count -- ;
}
System.out.print( " Done "  ) ;
```

count

4

OUTPUT

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)     TRUE
{
      System.out.println(count);


      count -- ;
}
System.out.print( " Done "  ) ;
```

count

4

OUTPUT

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);

      count -- ;
}
System.out.print( " Done "  ) ;
```

count

4

OUTPUT

```
4
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

3

OUTPUT

```
4
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)     TRUE
{
     System.out.println(count);


     count -- ;
}
System.out.print( " Done "  ) ;
```

count

3

OUTPUT

```
4
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

3

OUTPUT

```
4
3
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);

      count -- ;
}
System.out.print( " Done "  ) ;
```

count

2

OUTPUT

```
4
3
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)     TRUE
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

2

OUTPUT

```
4
3
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

2

OUTPUT

```
4
3
2
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);

      count -- ;
}
System.out.print( " Done "  ) ;
```

count

1

OUTPUT

```
4
3
2
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)     TRUE
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

1

OUTPUT

```
4
3
2
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
     System.out.println(count);

     count -- ;
}
System.out.print( " Done "  ) ;
```

count

1

OUTPUT

```
4
3
2
1
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);

      count -- ;
}
System.out.print( " Done "  ) ;
```

count

0

OUTPUT

```
4
3
2
1
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)     FALSE
{
     System.out.println(count);


     count -- ;
}
System.out.print( " Done "  ) ;
```

count

0

OUTPUT

```
4
3
2
1
```

The `while` Count-controlled Loop

```
int   count ;

count  =  4;

while (count > 0)
{
      System.out.println(count);


      count -- ;
}
System.out.print( " Done "  ) ;
```

count

```
0
```

OUTPUT

```
4
3
2
1
Done
```

Example `while` loop

```
// finds the first factor of 91, other than 1
int n = 91;
int factor = 2;  ←  Initialization
while (n % factor != 0) ← {  Test
    factor++;  ←  Update
}
System.out.println("First factor is " + factor);
// output:  First factor is 7
```

– `while` is better than `for` because we don't know how many times we will need to increment to find the factor.

Two Categories of Loops

count-controlled **(definite) loops**

repeat a specified number of times

sentinel-controlled (indefinite) loops

some condition within the loop body changes and this causes the repeating to stop

Sentinel values

- **sentinel**: A value that signals the end of user input.
  - **sentinel loop**: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for numbers until the user types -1, then outputs their sum and their average.
  - (In this case, -1 is the sentinel value.)

```
Enter a number (-1 to quit): 10
Enter a number (-1 to quit): 20
Enter a number (-1 to quit): 30
Enter a number (-1 to quit): -1
The sum is 60
The average is 20
```

Sentinel loop solution

```
Scanner console = new Scanner(System.in);
int sum = 0;
int count = 0;
// pull one prompt/read ("post") out of the loop
System.out.print("Enter a number (-1 to quit): ");
int number = console.nextInt();

while (number != -1) {
   sum = sum + number;     // moved to top of loop
   count++;
   System.out.print("Enter a number (-1 to quit): ");
   number = console.nextInt();
}

System.out.println("The sum is " + sum);
System.out.println("The average is " + sum/count);
```

"Forever" loop with break

- **`break` statement**: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - Such loops are often called *"forever" loops* because their header's boolean test is often changed to a trivial `true`.
- "forever" loop, general syntax:

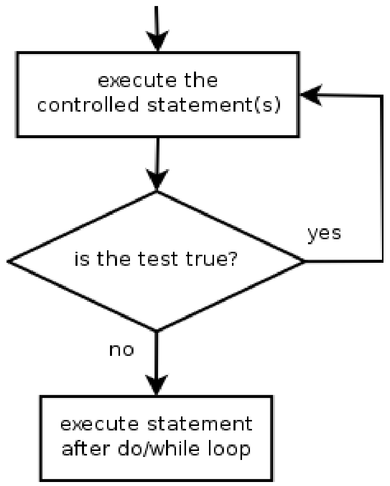
```
while (true) {
    <statement(s)> ;

    if (<condition>) {
        break;
    }
    <statement(s)> ;
}
```

- Can also be used to end the loop in the middle for any other condition other than the loop test

The `do/while` loop

- **`do/while` loop**: Performs its test at the *end* of each repetition.
  - Guarantees that the loop's `{ }` body will run at least once.

```
do {
    statement(s);
} while (test);
```


execute the controlled statement(s)
is the test true?
yes
no
execute statement after do/while loop

```
// Example: prompt until correct password is typed
String phrase;
do {
    System.out.print("Type your password: ");
    phrase = console.next();
} while (!phrase.equals("abracadabra"));
```

`do/while` question

- Rolls two dice until a sum of 7 is reached

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 tries!
```

do/while answer

```java
// Rolls two dice until a sum of 7 is reached.
public class Dice {
    public static void main(String[] args) {
        int tries = 0;
        int sum;

        do {
            int roll1 = 1 + (int) (Math.random() * 6); // one roll
            int roll2 = 1 + (int) (Math.random() * 6);
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        } while (sum != 7);

        System.out.println("You won after " + tries + " tries!");
    }
```

Do-While Loop vs. While Loop

- POST-TEST loop
- The looping condition is tested after executing the loop body.
- Loop body is always executed at least once.

- PRE-TEST loop
- The looping condition is tested before executing the loop body.
- Loop body may not be executed at all.

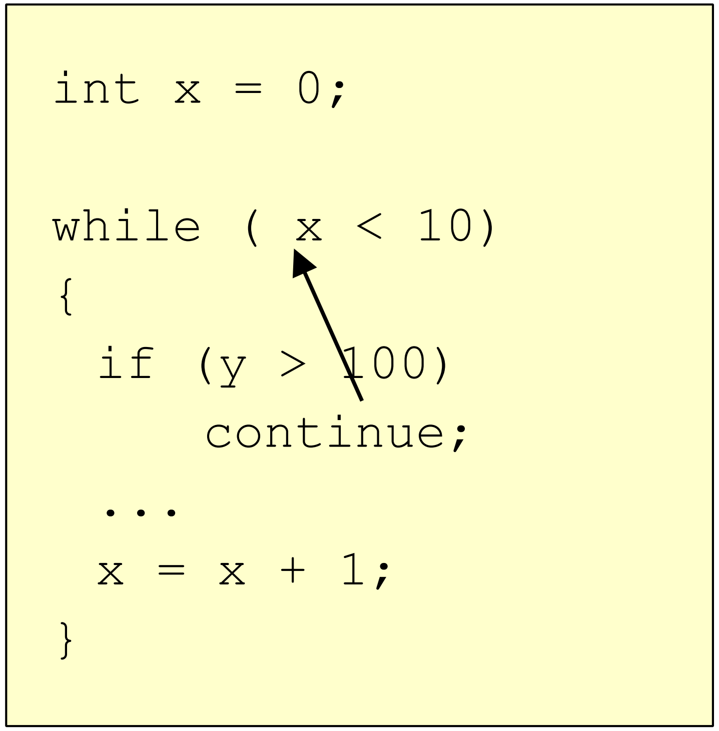
break statement (revisited)

- We previously saw the ***break*** statement used in switch statements and loops.
- The ***break*** statement will cause the flow of execution to break out of the current loop.
- If loops are nested, ***break*** will cause control to leave the inner-most loop.

```
int x = 0;

while ( x < 10)
{
  if (y > 100)
       break;
  [...]
  x = x + 1;
}
```

continue statement

- ***continue*** is similar to break.
- ***continue*** causes execution to go back to the loop test condition. If the test condition is true, the loop will be executed again. If not, the loop body is exited.



```
int x = 0;

while ( x < 10)
{
  if (y > 100)
       continue;
  ...
  x = x + 1;
}
```

Example - *continue*

```
// Compute the average of the odd numbers from 1 to n and
print them

public class ContinueEx {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in)
      double sum = 0.0;
      int n = console.nextInt();
      for (int i = 1; i <= n; i++) {
          if(i%2 == 0)
              continue;
          System.out.println(i);
          sum = sum + i;
        }
       System.out.println(sum / n);
    }
}
```