For Loops

# AP Computer Science

# 3 loops in Java

1. **`while`** loops
   - Used when you do not know how many times you are going to need to repeat
2. **`for`** loops
   - Used when you do know how many times you are going to repeat
3. **`do-while`** loops
   - Used rarely
   - Used whenever you need to be guaranteed the loop runs at least once

# Loops are interchangeable

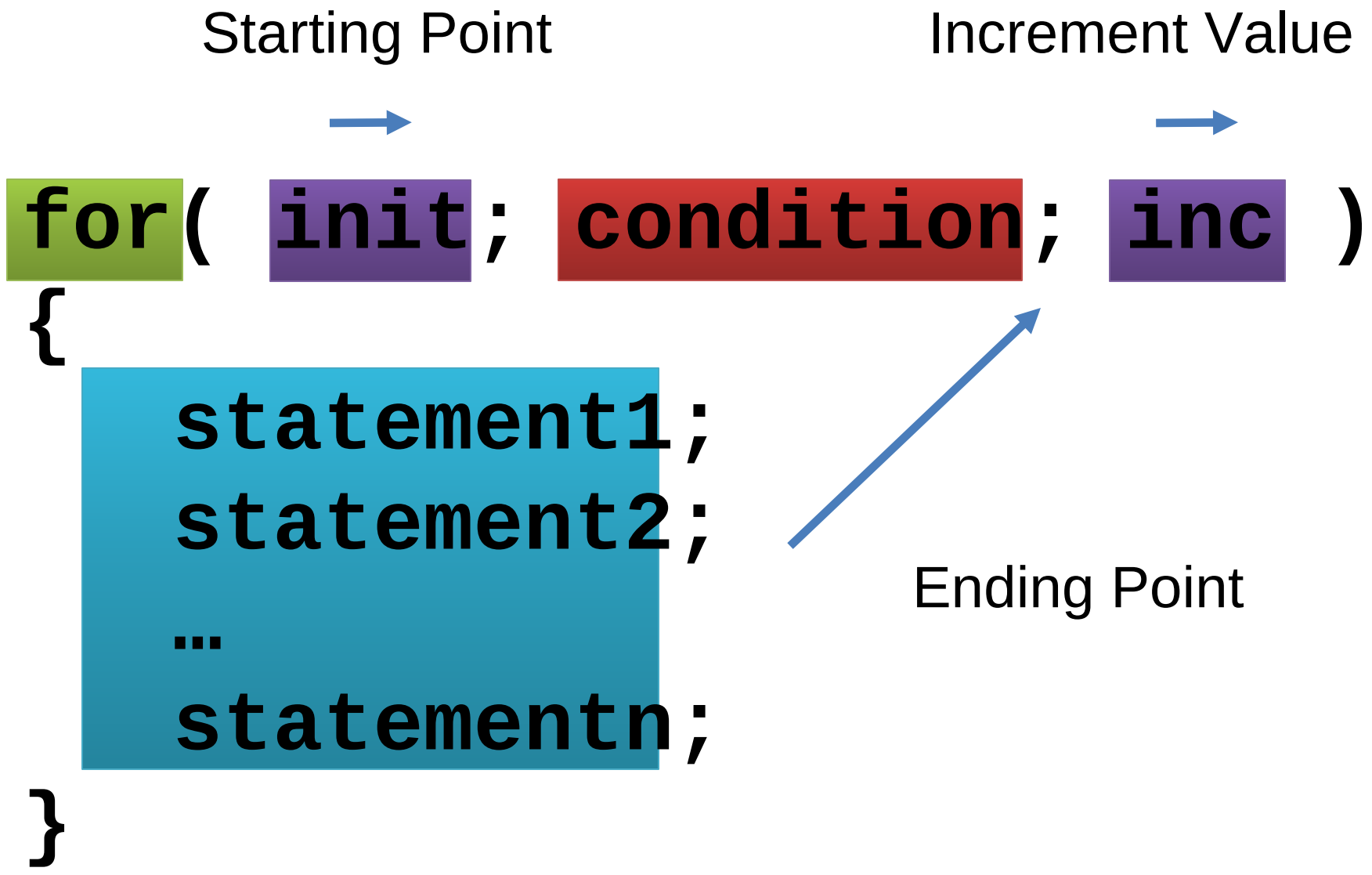
- Any problem requiring a loop can use any kind of loop
- The choice is to make things easier on the programmer
- Some loops are more convenient for certain kinds of problems

# for Loops

# Background on `for` loops

- **`for`** loops are great when you know how many times a loop will run
- They are the most commonly used of all loops
- They are perfect for any task that needs to run, say, 100 times
- A **`for`** loop has 3 parts in its header:
  1. Initialization
  2. Condition
  3. Increment

# Anatomy of a for loop



# `for` loop example

- Print the numbers from 1 to 10 (again)
- Remember how this was done with **`while`**:

```
int i = 1;

while( i <= 10 )
{
  System.out.println(i);
  i++;
}
```

# `for` loop example

- A **`for`** loop is specifically designed for this sort of thing:

```
for( int i = 1; i <= 10; i++ )
{
  System.out.println(i);
}
```

- The initialization and increment are built-in

# Counting

- Counting by 2

```
for(int c = 0; c < 10; c = c + 2)
{
  System.out.println(c);
}
```

| Output |
| --- |
| 0 |
| 2 |
| 4 |
| 6 |
| 8 |

- Counting by 3

```
for(int c = 0; c < 10; c = c + 3)
{
  System.out.println(c);
}
```

| Output |
| --- |
| 0 |
| 3 |
| 6 |
| 9 |

# Summing Numbers

- Solution:

```
int sum = 0;
for(int c = 4; c > 0; c--)
{
  sum += c;
}
System.out.println(sum);
```

| Output |
| --- |
| 10 |

- Notice the start/stop conditions!

# Finding the Average

- We could also find the average:

```
double sum = 0;
Scanner in = new Scanner( System.in );
System.out.println("How many numbers? ");
int prompt = in.nextInt();
for(int count = prompt; count > 0; count-- )
{
  System.out.print("Enter number: ");
  int i = in.nextInt();
  sum += i;
}
System.out.println("Avg: " + (sum/prompt));
 //Why did we use prompt instead of count?
```

# Common Loop Errors

# Infinite `for` loops

- Infinite **`for`** loops are unusual, but possible:

```
for( ; ; )
  System.out.println("Hey!");
```

- This situation is more likely:

```
for( int i = 0; i < 10; i++ )
{
  System.out.println(i);
  //other code
  i--; //whoops
}
```

# (Almost) infinite loops

- Overflow and underflow will make some badly written loops **eventually** terminate

```
int i;
//whoops, should have been i++
for( i = 1; i <= 40; i-- )
{
  System.out.println(i);
}
```

# Fencepost errors

- Being off by one is a very common error

```
int i;
//runs 39 times
for( i = 1; i < 40; i++ )
{
  System.out.println(i);
}
```

# Skipping loops entirely

- If the condition is not **`true`** to begin with, the loop will be skipped

```
//oops, should be <=
for( int i = 1; i >= 40; i++ )
{
  System.out.println(i);
}
```

# Misplaced semicolon

- A misplaced semicolon can cause an empty loop body to be executed

```
int i;
//semicolon is wrong
for( i = 1; i <= 40; i++ );
{
  System.out.println(i);
}
```

- Everything looks good and the loop even terminates
- But, only one number will be printed: 41

# do-while Loops

# No one really uses `do-while` loops

- They work like **`while`** loops
- The only difference is they are guaranteed to execute at least once
- Unlike a **`while`** loop, the condition is not checked the first time you go into the loop
- Sometimes this is especially useful for getting input from the user

# Anatomy of a do-while loop

```
do
{
    statement1;
    statement2;
    …
    statementn;
} while( condition );
```

# Menu with a `do-while`

- The do-while loop is rarely used, but certain kinds of input are well suited to a **`do-while`**
- For example, a program that gives a menu of options
  1. Add two numbers
  2. Subtract two numbers
  3. Multiply two numbers
  4. Quit

# do-while Example

Please open **DoWhile.java**