AP Computer Science A: `while` *Loops*

CodeHS

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

```
//Step One: Write method signature

//Step Two: Create Scanner

//Step Three: Ask user for two numbers

//Step Four: Print the sum
```

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

```
public static void sum()
{
//Step Two: Create Scanner

//Step Three: Ask user for two numbers

//Step Four: Print the sum
```

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

```
public static void sum()
{
    Scanner input = new Scanner(System.in);

//Step Three: Ask user for two numbers

//Step Four: Print the sum
```

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

```
public static void sum()
{
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a number: ");
    double x = input.nextDouble();
    System.out.println("Enter another number: ");
    double y = input.nextDouble();
//Step Four: Print the sum
```

Calculator Class

Let's create a `Calculator` Class with a static method `sum()` that asks a user for two numbers and prints the sum of those two numbers as a double.

```
public static void sum()
{
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a number: ");
    double x = input.nextDouble();
    System.out.println("Enter another number: ");
    double y = input.nextDouble();
    System.out.println("The sum of " + x + "," + y + " is " + (x+y));

}
```

public static void sum()

AP CSA (Java (main))

SAVE

```java
import java.util.Scanner;

class Calculator
{
    public static void sum()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a number: ");
        double x = input.nextDouble();
        System.out.println("Enter a second number: ");
        double y = input.nextDouble();
        System.out.println("The sum of " +x +" and " +y+ " is: " + (x + y));
    }
}
```

RUN CODE | DOCS | MORE

RUN CODE STOP DEBUG

sum Improved?

What if we wanted to get **10 numbers** from a user, and compute the sum of all 10 numbers?

Calculator Class

Adding more numbers to sum makes this method harder to implement:

```
public static void sum()
{
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a number: ");
    double first = input.nextDouble();
    System.out.println("Enter another number: ");
    double second = input.nextDouble();
    System.out.println("The sum of " + x + "," + y + " is " + (x+y));

}
```

Calculator Class

Adding more numbers to sum makes this method harder to implement:

```
public static void sum()
{
    Scanner input = new Scanner(System.in);
    System.out.println("Enter a number: ");
    double first = input.nextDouble();
    System.out.println("Enter another number: ");
    double second = input.nextDouble();
    System.out.println("The sum of " + x + "," + y + " is " + (x+y));

}
```

If we wanted to sum 10 numbers, we would need to include 9 more variables and print statements!

`while` Loops

We can avoid writing repetitive code by using **`while` loops** in our programs.

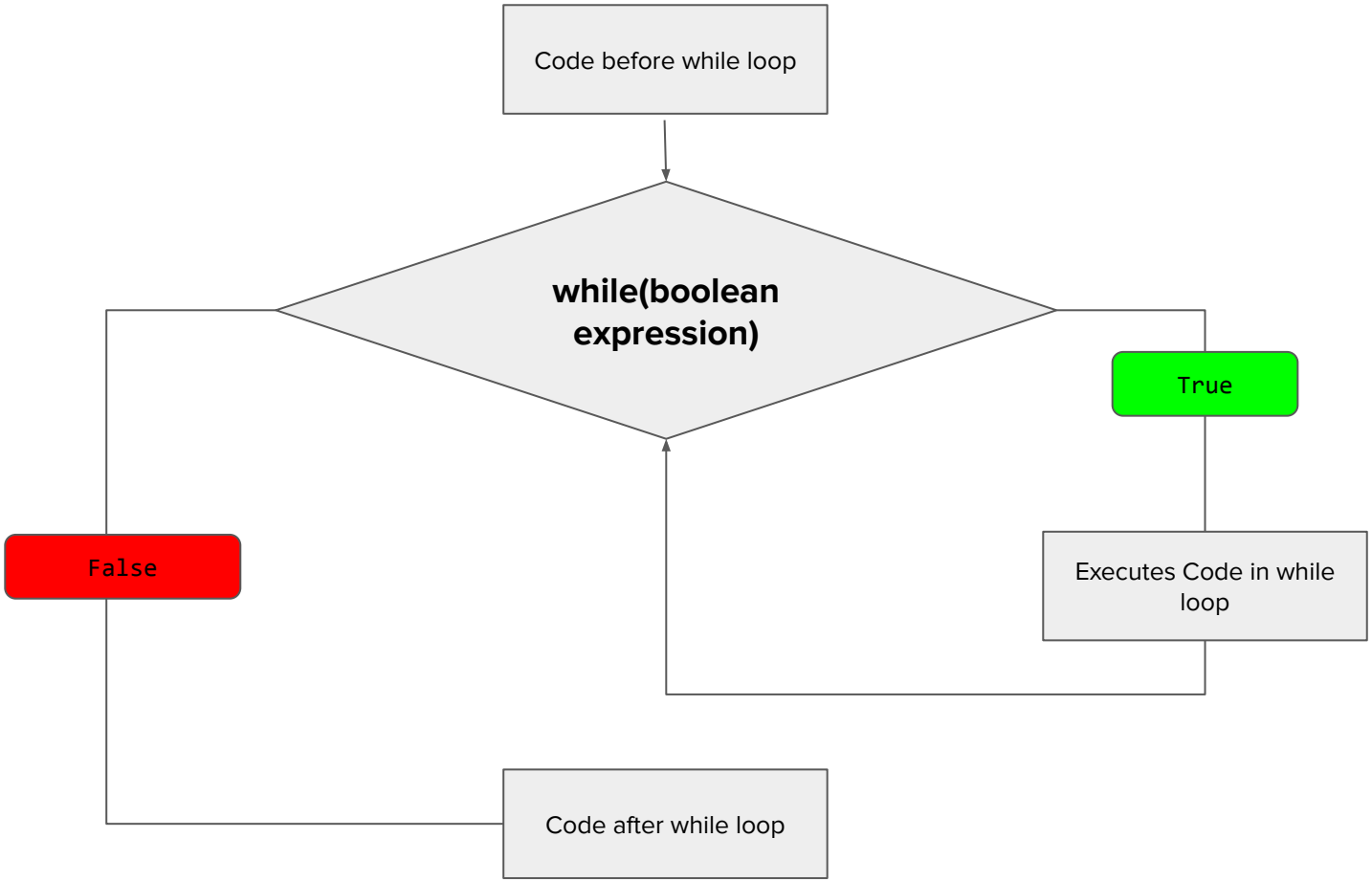
`while` Loops

`while` loops allow us to repeat a set of statements **until a condition is met.**

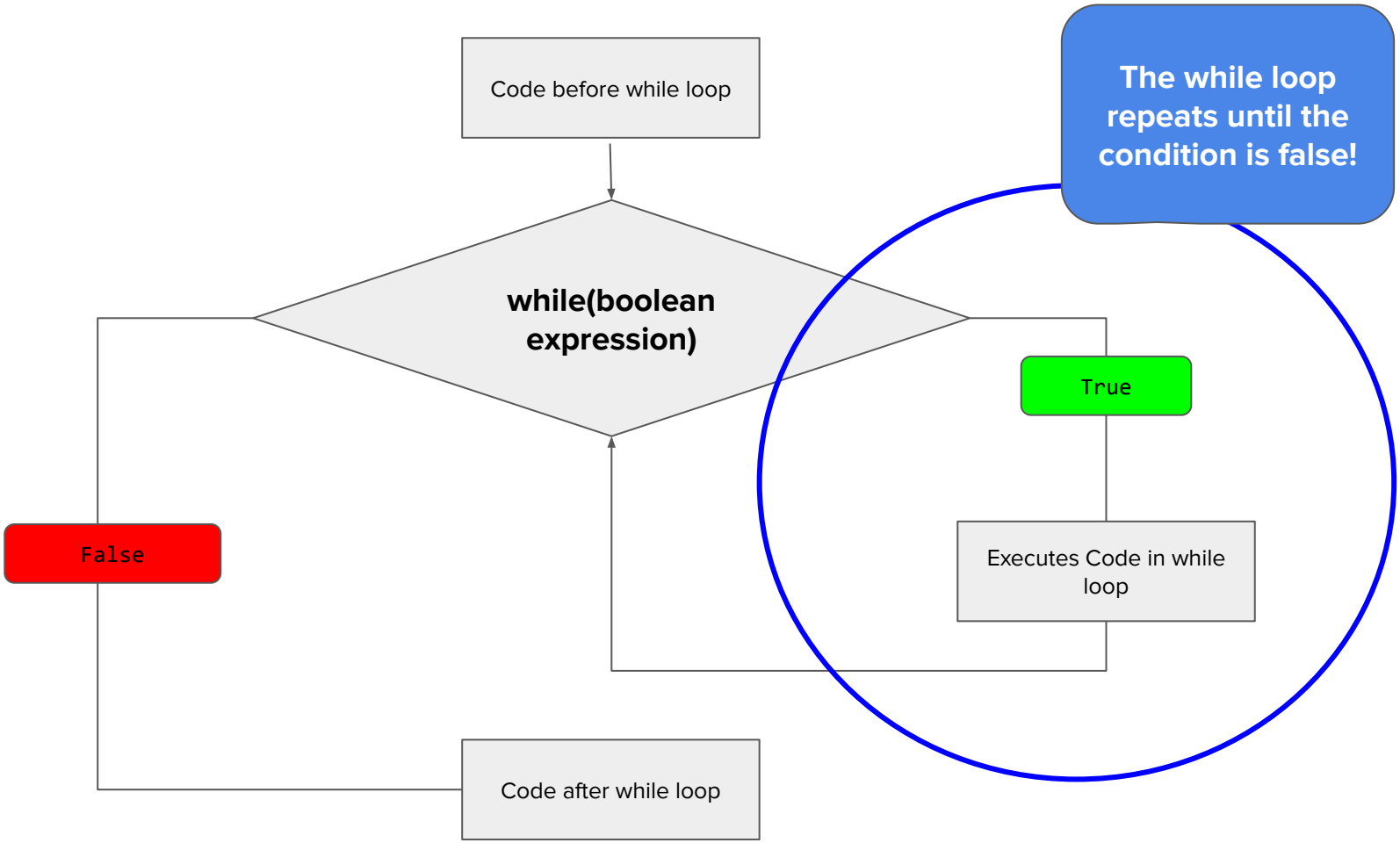
while Loops

```
while(boolean expression)
{
   //will execute if boolean is true, and until
   the boolean expression is false
}
```

`while` Loop Flowchart



`while` Loop Flowchart



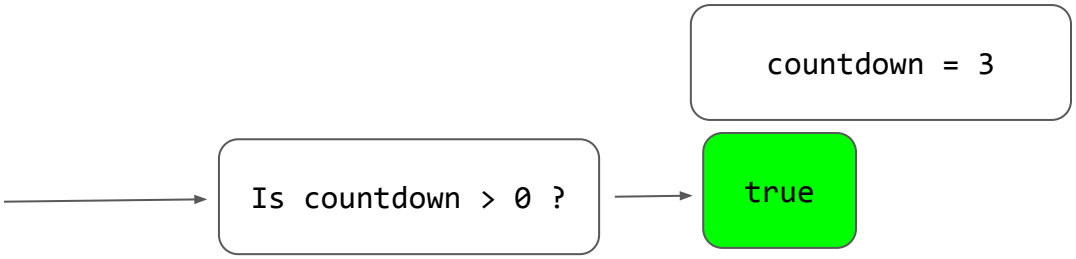
`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 3
```

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```

countdown = 3

3

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 2
```

`while` Loop Example

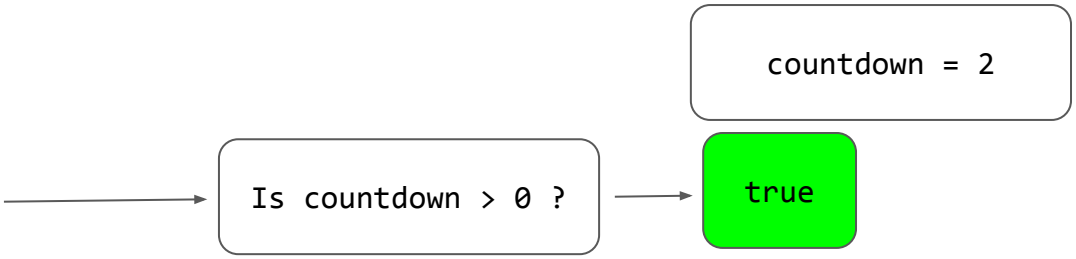
```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 2
```

At the end of the while loop, **the program goes back to the initial while loop condition!**

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```

countdown = 2

2

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 1
```

`while` Loop Example

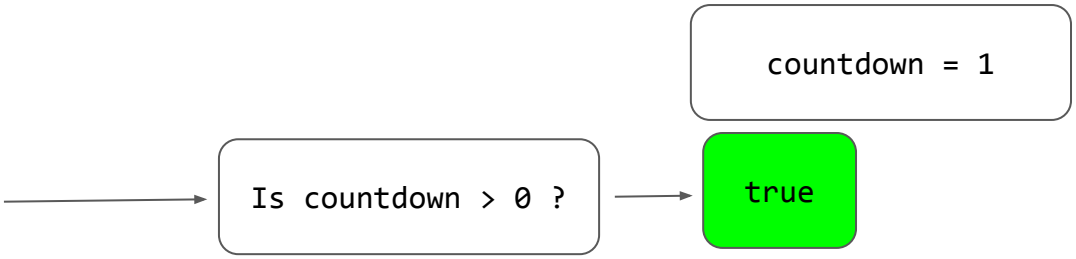
```
int countdown = 3;

while(countdown > 0)
{
    System.out.println(countdown);
    countdown--;
}
System.out.println("Countdown Complete");
```

`countdown = 1`

At the end of the while loop, **the program goes back to the initial while loop condition!**

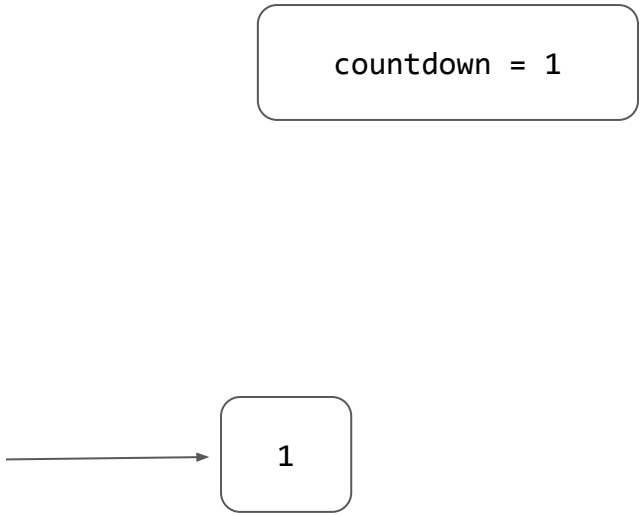
`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 0
```

`while` Loop Example

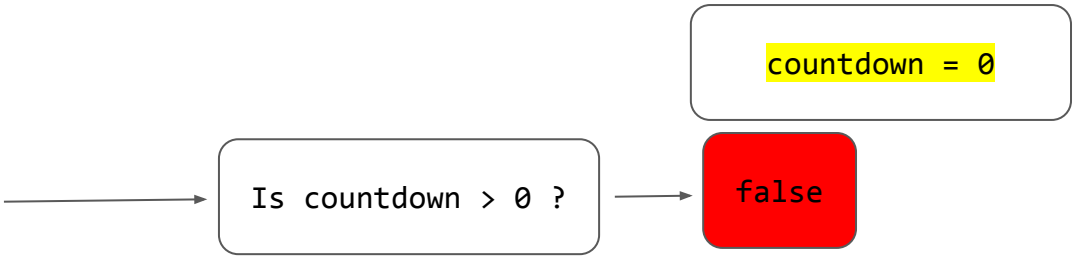
```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```

`countdown = 0`

At the end of the while loop, **the program goes back to the initial while loop condition!**

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

```
countdown = 0
```

`while` Loop Example

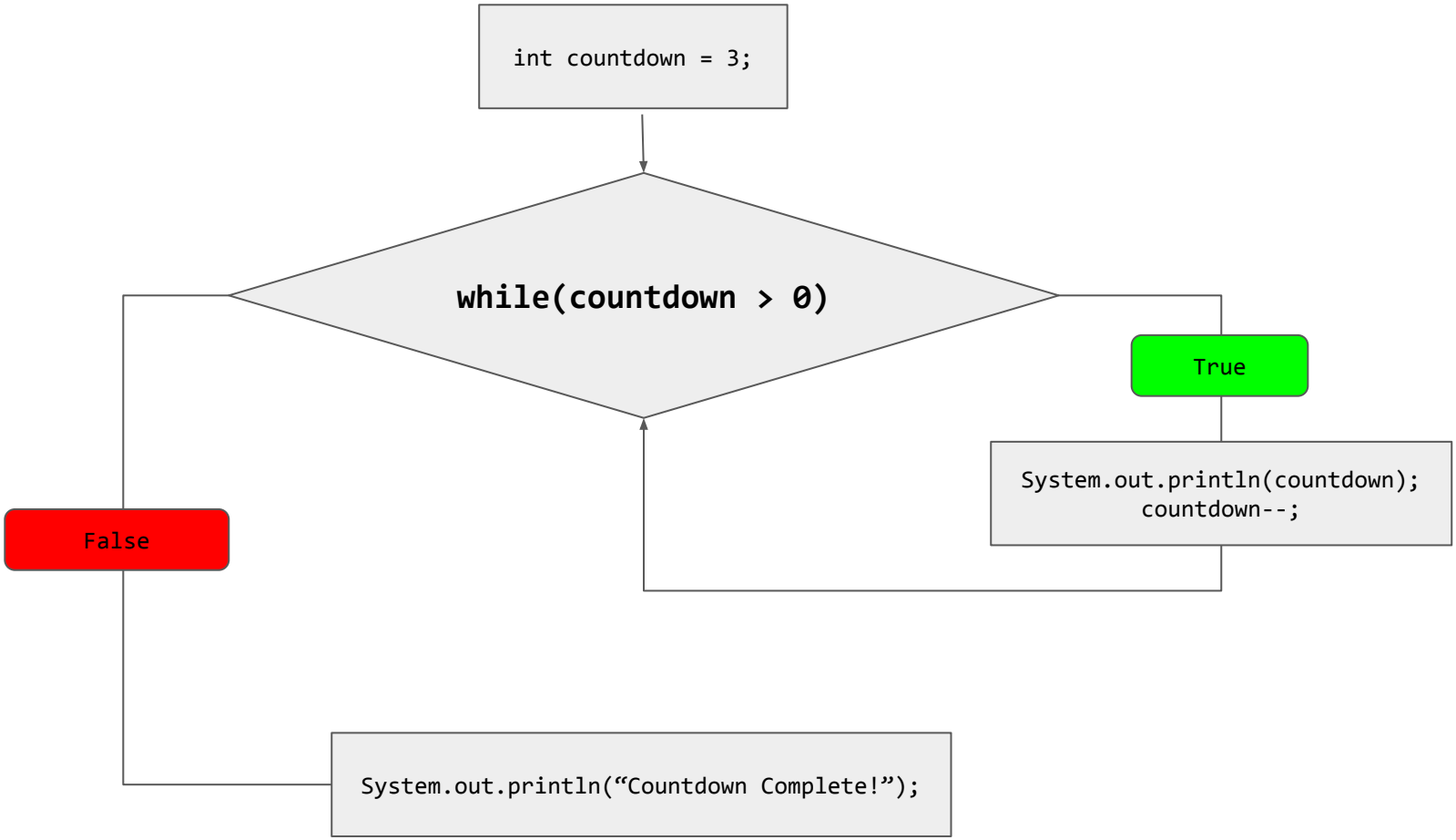
AP CSA (Java (main))

```java
class MyProgram {
    public static void main(String[] args)
    {
        int countdown = 3;
        while(countdown > 0)
        {
            System.out.println(countdown);
            countdown--;
        }
        System.out.println("Countdown Complete!");

    }
}
```

RUN CODE | DOCS | MORE

STOP DEBUG

while Loop Flowchart



`while` Loop Example

```
int countdown = 0;

while(countdown > 0)

{

   System.out.println(countdown);

   countdown--;

}

System.out.println("Countdown Complete");
```

If our initial condition doesn't pass the boolean expression, the while loop will never execute

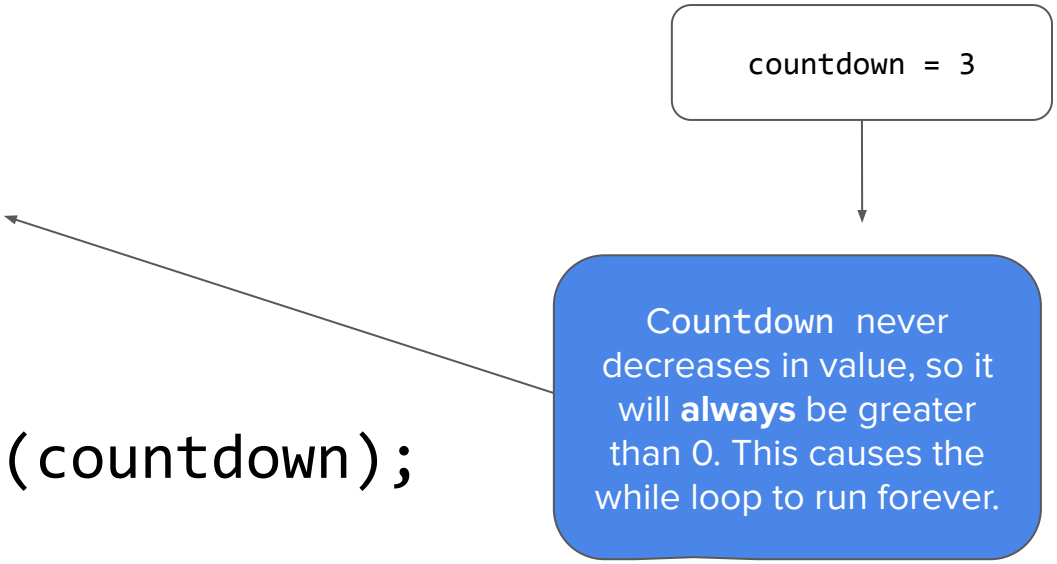
`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

    System.out.println(countdown);

    countdown--;

}

System.out.println("Countdown Complete");
```

This while loop only works because of the countdown--. If this was removed, the program would **run forever!**

`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

}

System.out.println("Countdown Complete");
```



`while` Loop Example

```
int countdown = 3;

while(countdown > 0)

{

   System.out.println(countdown);

}

System.out.println("Countdown Complete");
```

This is called an **infinite loop.** If the condition of the while loop never changes to false, the program will crash.

`while` Loop Example

AP CSA (Java (main))

SAVE

RUN CODE DOCS MORE

```java
class MyProgram
{
    public static void main(String[] args)
    {
       int countdown = 3;
        while(countdown > 0)
        {
            System.out.println(countdown);
        }
        System.out.println("Countdown Complete");
    }
}
```

RUN CODE STOP DEBUG

Avoiding Infinite Loops

Fix this infinite loop!

```
//This program counts from 0-100 by 5

int x = 0;
while(x < 100)
{
    System.out.println(x);

}
```

Avoiding Infinite Loops

Fix this infinite loop!

```
//This program counts from 0-100 by 5

int x = 0;
while(x < 100)
{
    System.out.println(x);
    x = x + 5;
}
```

We need to increase the value of x so the program ends once x gets to 100.

Avoiding Infinite Loops

Add a condition to the `while` loop!

```
//This creates a 8 length password of the letter “a”


String password = “”;
while()
{
    password+= “a”;
}
System.println(password);
```

Avoiding Infinite Loops

Add a condition to the `while` loop!

```
//This creates a 8 length password of the letter “a”


String password = “”;
while(password.length() < 8)
{
    password+= “a”;
}
System.println(password);
```

The program will stop once the length of the password is 8!

Infinite Loops

We can terminate while loops by adding a **break** statement.

Break

Break statements allow you to “break out” of the while loop and continue the program execution.

Infinite Loop

Writing while(true) will make a program run infinitely:

```
while(true)

{





}
```

Break Statements

Break statements will halt the execution of a while loop.

```
int counter = 0;
while(true)
{
    if(counter == 5)
    {
        break;
    }
    counter++;
}
```

Break Statements

Break statements will halt the execution of a while loop.

```
int counter = 0;
while(true)
{
    if(counter == 5)
    {
        break;
    }
    counter++;
}
```

While(true) will cause the loop to run forever, because the condition is always true.

Break Statements

Break statements will halt the execution of a while loop.

```
int counter = 0;
while(true)
{
    if(counter == 5)
    {
        break;
    }
    counter++;
}
```

The break statement allows us to exit the while loop and continue executing the program.

Halting an Infinite Loop

Break statements will halt the execution of a while loop.

AP CSA (Java (main))

SAVE

```
1  class MyProgram
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("This code executes because it's before the infinte loop!");
6          int counter = 0;
7          while(true)
8          {
9              counter++;
10         }
11         System.out.println("The loop is over!");
12     }
13
14 }
15
```

RUN CODE | DOCS | MORE

RUN CODE STOP DEBUG

```
MyProgram.java:11: error: unreachable statement
        System.out.println("The loop is over!");
        ^
1 error
```

Errors:

MyProgram.java: Line 11: unreachable statement

Infinite Loop

```
int counter = 0;
while(true)
{
    if(counter == 5)
    {
        break;
    }
    counter++;
}
```

If we removed this, the while loop would still be stuck in an infinite loop because the break statement will never execute if counter stays at 0

Return

Another way to halt while loops is by using the **return** keyword.

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("weakPassword"))
     {
          break;
     }
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

This line of code **will execute** because the break statement ends the while loop and continues to the next line of code following the while loop

Return

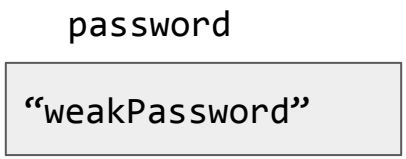
```
String password = “weakPassword”;

while(true)
{
     if(password.equals("weakPassword"))
     {
          return;
     }
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

This line of code **will not** execute because the return statement exits the method or constructor regardless of what code follows the return statement or the while loop

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

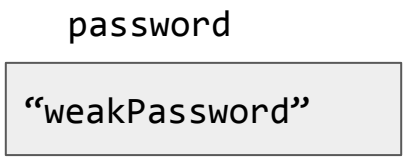
password

“weakPassword”

Break vs Return

Break

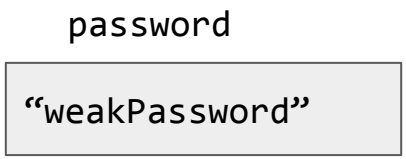
```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

password

“strongPassword”

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

password

“strongPassword”

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

password

“strongPassword”

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

password

“strongPassword”

Break vs Return

Break

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          break;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

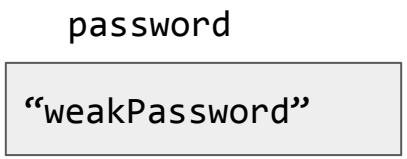
password

“strongPassword”

Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

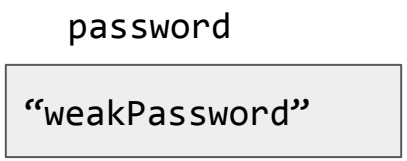
password

“weakPassword”

Break vs Return

Return

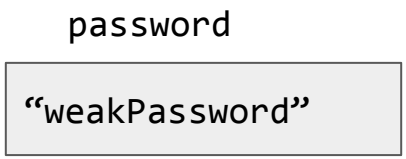
```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```



Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

password

“strongPassword”

Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

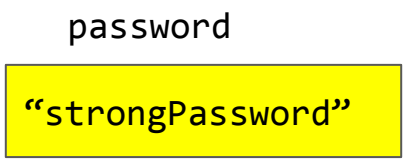
password

“strongPassword”

Break vs Return

Return

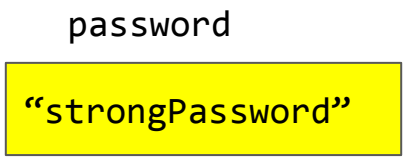
```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```

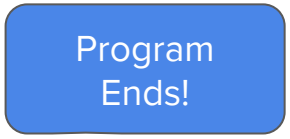


Break vs Return

Return

```
String password = “weakPassword”;

while(true)
{
     if(password.equals("strongPassword"))
     {
          return “Success!”;
     }
     System.out.println("Weak password, try again: ");
     password = input.nextLine();
}
System.out.println(“Next line of code”);
```





Let’s return to our initial Calculator class!

Improved sum()

Specifications for `sum()`:

`sum()` asks the user to specify how many numbers they want to sum together. Then it asks the user to input one number at a time, adding to the running sum total each time, and eventually returning the correct sum.

sum() method

```
//Step One: Ask the user for numbers involved in the sum

//Step Two: Create a counter

//Step Three: Create a running sum total

//Step Four: Create a while loop that runs until counter hits
requested numbers

    //Step Five: Ask user for number

    //Step Six: Add number to running sum

    //Step Seven: Add to counter

//Step Eight: Return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
//Step Two: Create a counter

//Step Three: Create a running sum total

//Step Four: Create a while loop that runs until counter hits
requested numbers

    //Step Five: Ask user for number

    //Step Six: add number to running sum

    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;

//Step Three: Create a running sum total

//Step Four: Create a while loop that runs until counter hits
requested numbers

    //Step Five: Ask user for number

    //Step Six: add number to running sum

    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
//Step Four: Create a while loop that runs until counter hits
requested numbers

    //Step Five: Ask user for number

    //Step Six: add number to running sum

    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
while(counter < totalNumbers)
{
    //Step Five: Ask user for number

    //Step Six: add number to running sum

    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
while(counter < totalNumbers)
{
    System.out.println("Enter a number to add:");
    int newNumber = input.nextInt();
    //Step Six: add number to running sum

    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
while(counter < totalNumbers)
{
    System.out.println("Enter a number to add:");
    int newNumber = input.nextInt();
    sum += newNumber;
    //Step Seven: add to counter

//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
while(counter < totalNumbers)
{
    System.out.println("Enter a number to add:");
    int newNumber = input.nextInt();
    sum += newNumber;
    counter++;
//Step Eight: return sum
```

sum() method

```
Scanner input = new Scanner(System.in);
System.out.println("How many numbers are you adding?");
int totalNumbers = input.nextInt();
int counter = 0;
int sum = 0;
while(counter < totalNumbers)
{
    System.out.println("Enter a number to add:");
    int newNumber = input.nextInt();
    sum += newNumber;
    counter++;
}
return sum;
```

sum( ) method

AP CSA (Java (main))

SAVE

```java
import java.util.Scanner;

class Calculator
{
    public static int sum()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("How many numbers are you adding?");
        int totalNumbers = input.nextInt();
        int counter = 0;
        int sum = 0;
        while(counter < totalNumbers)
        {
            System.out.println("Enter a number to add:");
            int newNumber = input.nextInt();
            sum += newNumber;
            counter++;
        }
        return sum;
    }
}
```

RUN CODE DOCS MORE

RUN CODE STOP DEBUG

Now It's Your Turn!

CodeHS

Concepts Learned this Lesson

Term	Definition

While loops	`while(boolean expression)`
	`{`
	`  //code executes until false`
	`}`

Infinite loops	Occurs when the expression in a while loop never evaluates to false. The program continues to run infinitely.
break	Breaks out of a while loop and executes statements that immediately follow while loop.
return	Keyword used in methods to return a value back to the initial program that called the method.

Standards Covered

- (LO) CON-2.C Represent iterative processes using a while loop.
- (EK) CON-2.C.1 Iteration statements change the flow of control by repeating a set of statements zero or more times until a condition is met.
- (EK) CON-2.C.2 In loops, the Boolean expression is evaluated before each iteration of the loop body, including the first. When the expression evaluates to true, the loop body is executed. This continues until the expression evaluates to false, whereupon the iteration ceases.
- (EK) CON-2.C.3 A loop is an infinite loop when the Boolean expression always evaluates to true.
- (EK) CON-2.C.4 If the Boolean expression evaluates to false initially, the loop body is not executed at all.
- (EK) CON-2.C.5 Executing a return statement inside an iteration statement will halt the loop and exit the method or constructor.
- (LO) CON-2.D For algorithms in the context of a particular specification that does not require the use of traversals:
  - Identify standard algorithms.
  - Modify standard algorithms.
  - Develop an algorithm.
- (EK) CON-2.D.1 There are standard algorithms to:
  - Identify if an integer is or is not evenly divisible by another integer
  - Identify the individual digits in an integer
  - Determine the frequency with which a specific criterion is met
- (EK) CON-2.D.2 There are standard algorithms to:
  - Determine a minimum or maximum value
  - Compute a sum, average, or mode