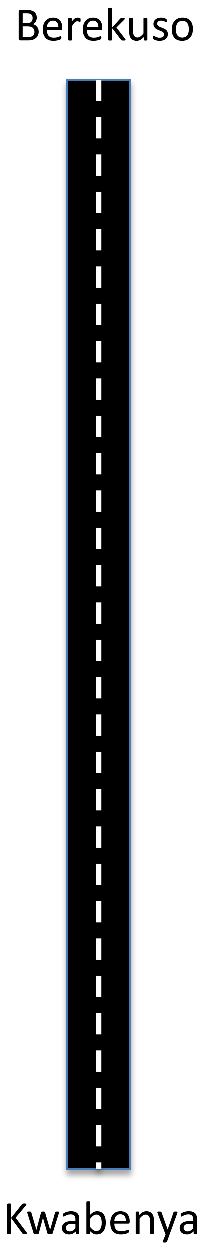
Flow of Control

Controlling the Order in which a Program Performs Actions

ASHESI

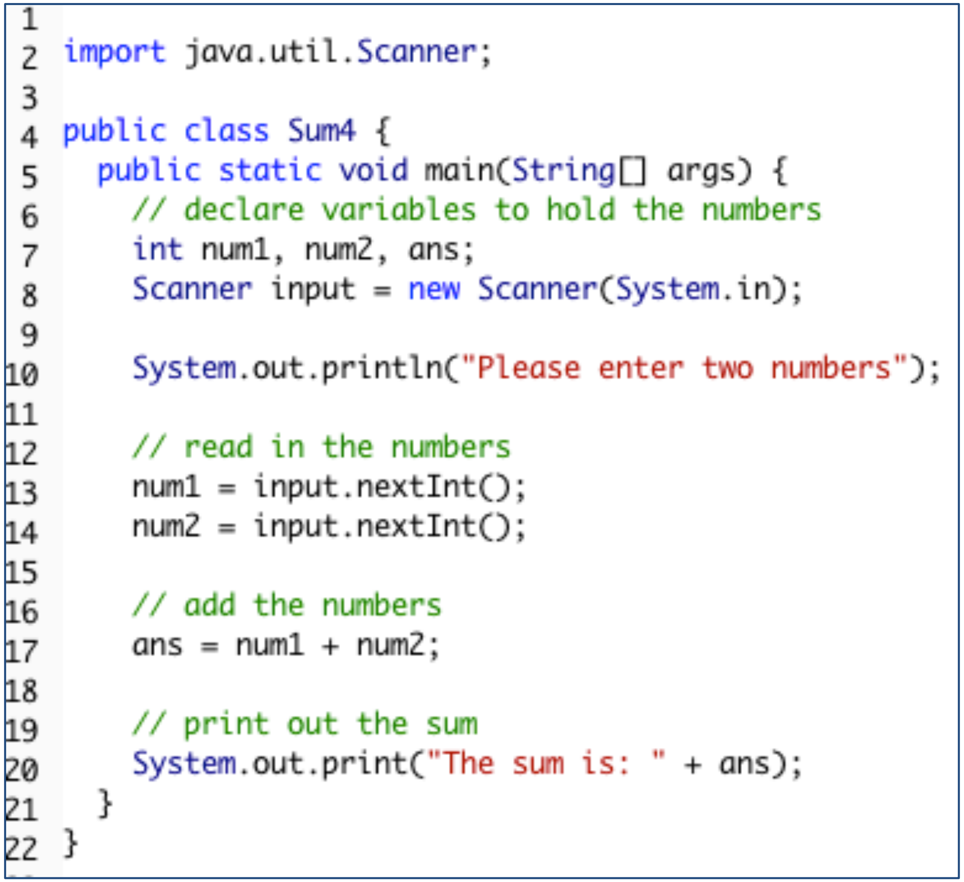
Flow of Control

- In simple programs, actions are taken in the order in which they are written down
- This is not very interesting – it's like a road that has no branches

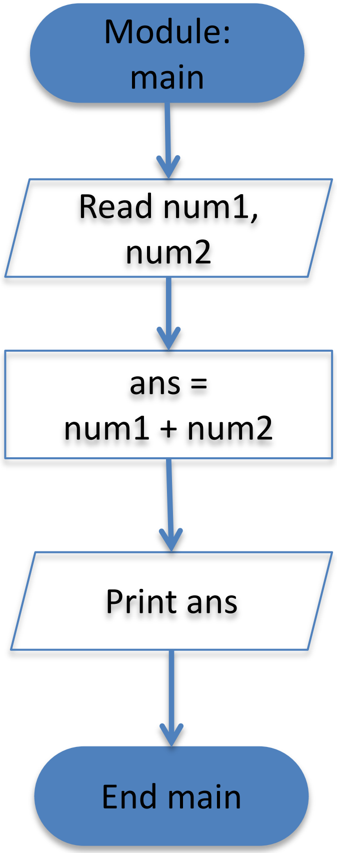


ASHESI

Flowchart for a simple program



```java

import java.util.Scanner;

public class Sum4 {
  public static void main(String[] args) {
    // declare variables to hold the numbers
    int num1, num2, ans;
    Scanner input = new Scanner(System.in);

    System.out.println("Please enter two numbers");

    // read in the numbers
    num1 = input.nextInt();
    num2 = input.nextInt();

    // add the numbers
    ans = num1 + num2;

    // print out the sum
    System.out.print("The sum is: " + ans);
  }
}
```



Module: main

Read num1, num2

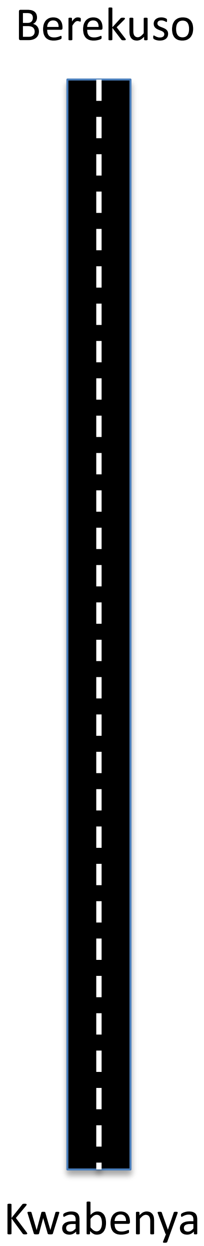
ans = num1 + num2

Print ans

End main

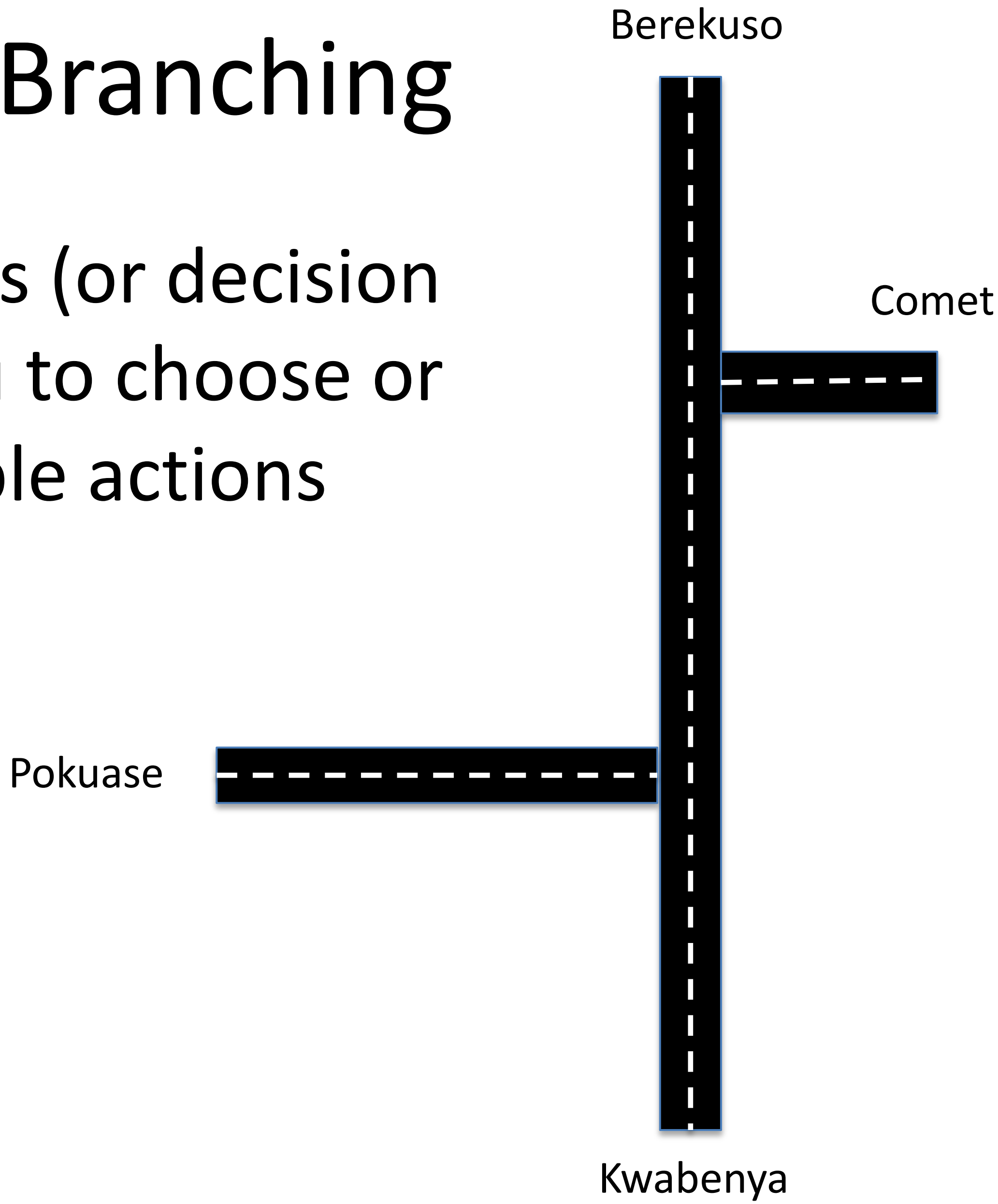
Flow of Control

- There are two ways to change the order in which a program performs actions:
  - Branching
  - Looping



Flow of Control: Branching

- Branching statements (or decision structures) allow you to choose or decide among possible actions



ASHESI

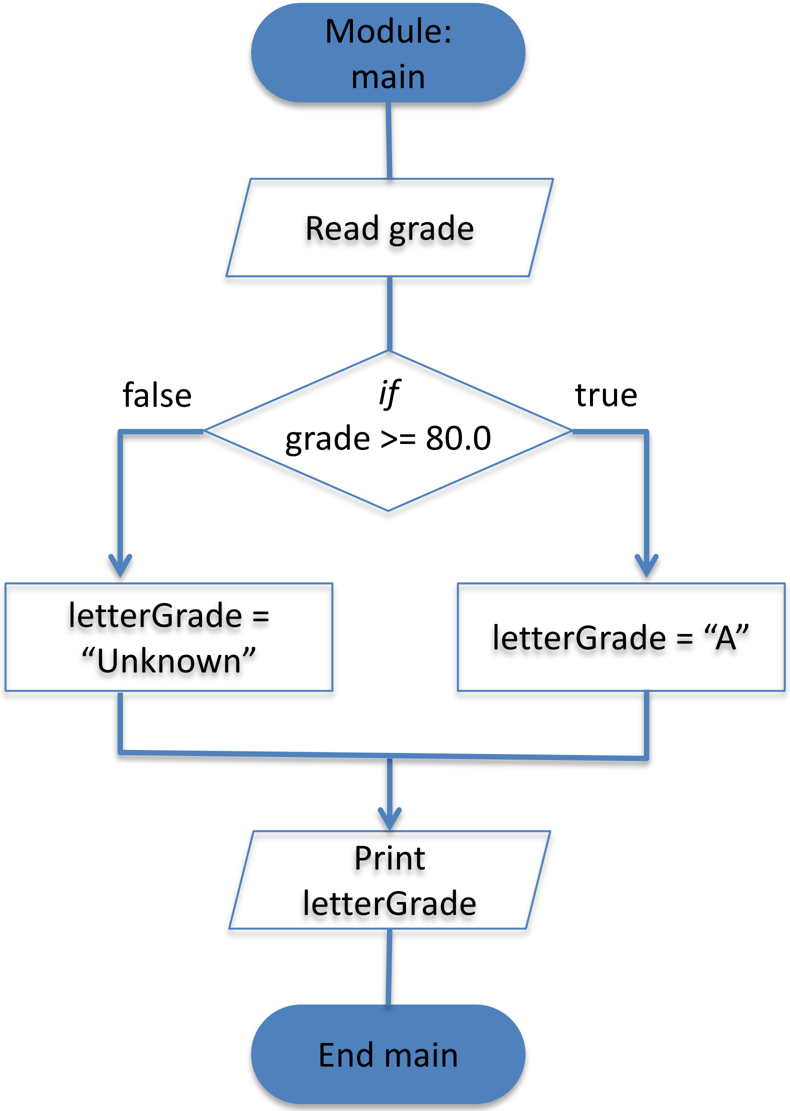
Branching: if … else

- if … else lets you implement conditional behavior in your program

```
if (boolean_expression) {
    // what to do if true
}
else {
    // what to do if false
}
```

if … else example

```
double grade;
String letterGrade;
Scanner input = new Scanner(System.in);

System.out.print(“Enter % grade: “);
grade = input.nextDouble();

if (grade >= 80.0) {
    letterGrade = “A”;
}

else {
    letterGrade = “Unknown”;
}

System.out.println(“Letter grade is: “
                  + letterGrade);
```



Module: main

Read grade

if grade >= 80.0

false

true

letterGrade = “Unknown”

letterGrade = “A”

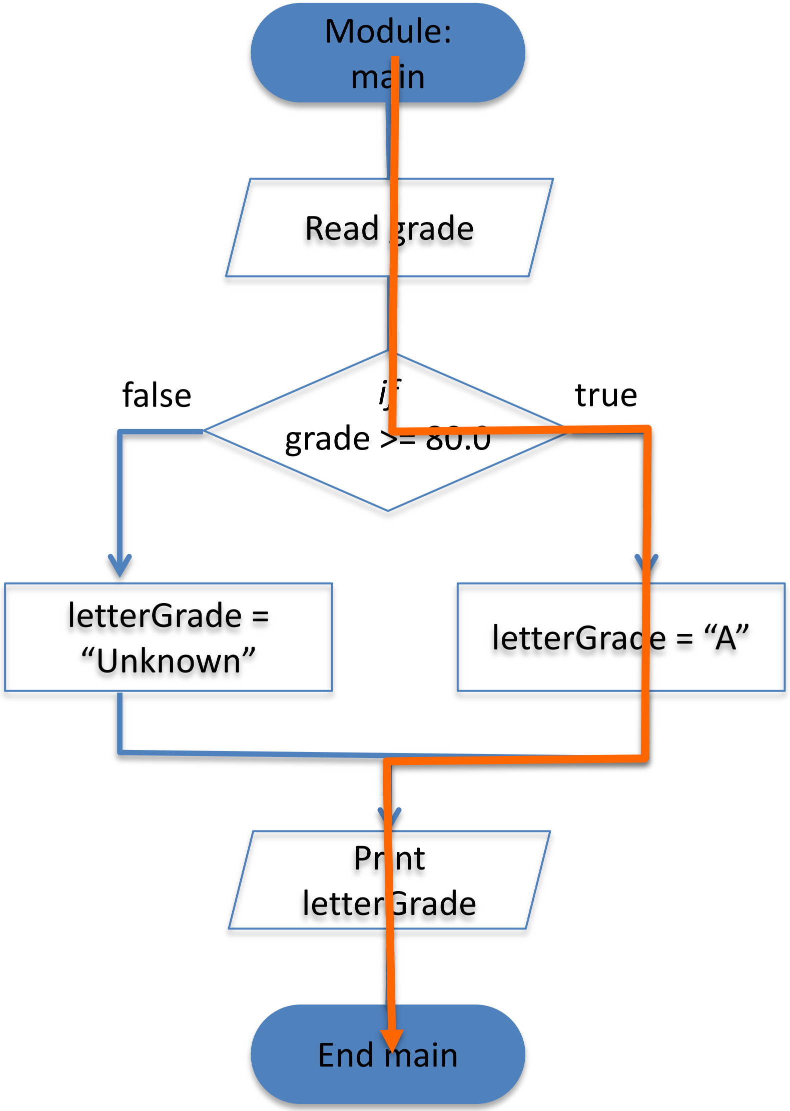
Print letterGrade

End main

if … else example

```
double grade;
String letterGrade;
Scanner input = new Scanner(System.in);

System.out.print("Enter % grade: ");
grade = input.nextDouble();

if (grade >= 80.0) {
     letterGrade = "A";
}

else {
     letterGrade = "Unknown";
}

System.out.println("Letter grade is: "
                   + letterGrade);
```

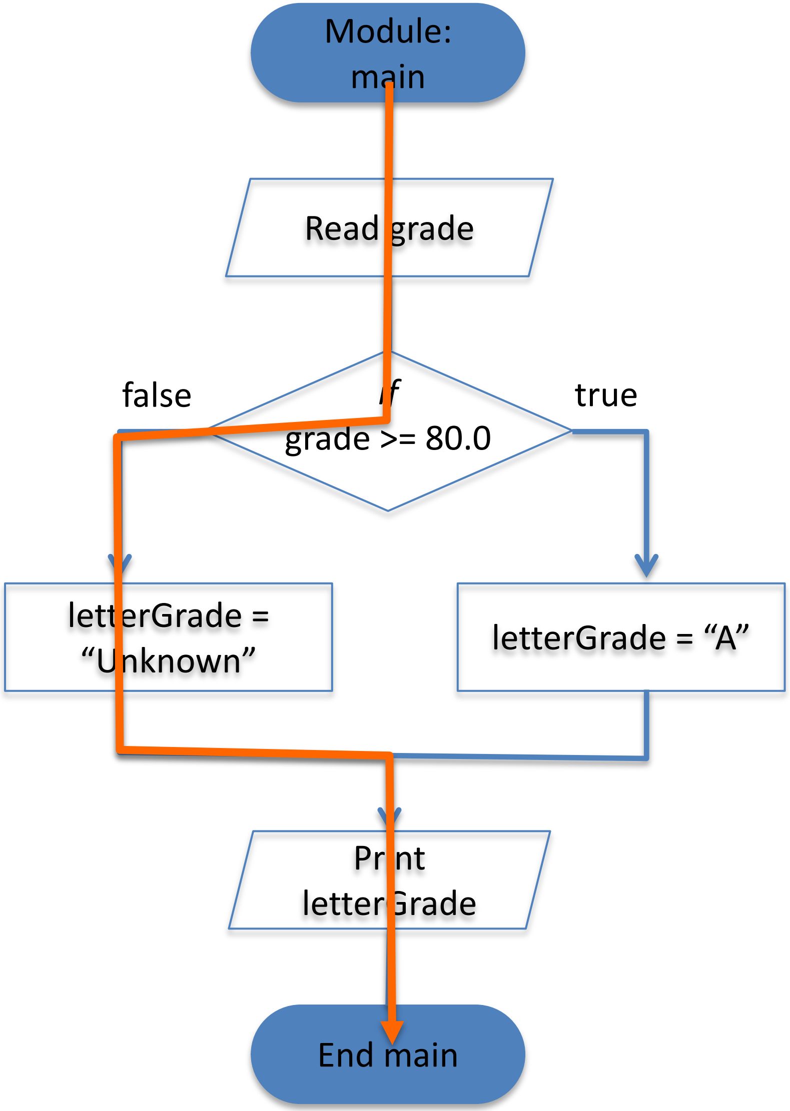
Path through program with test input: grade = 82.0



if … else example

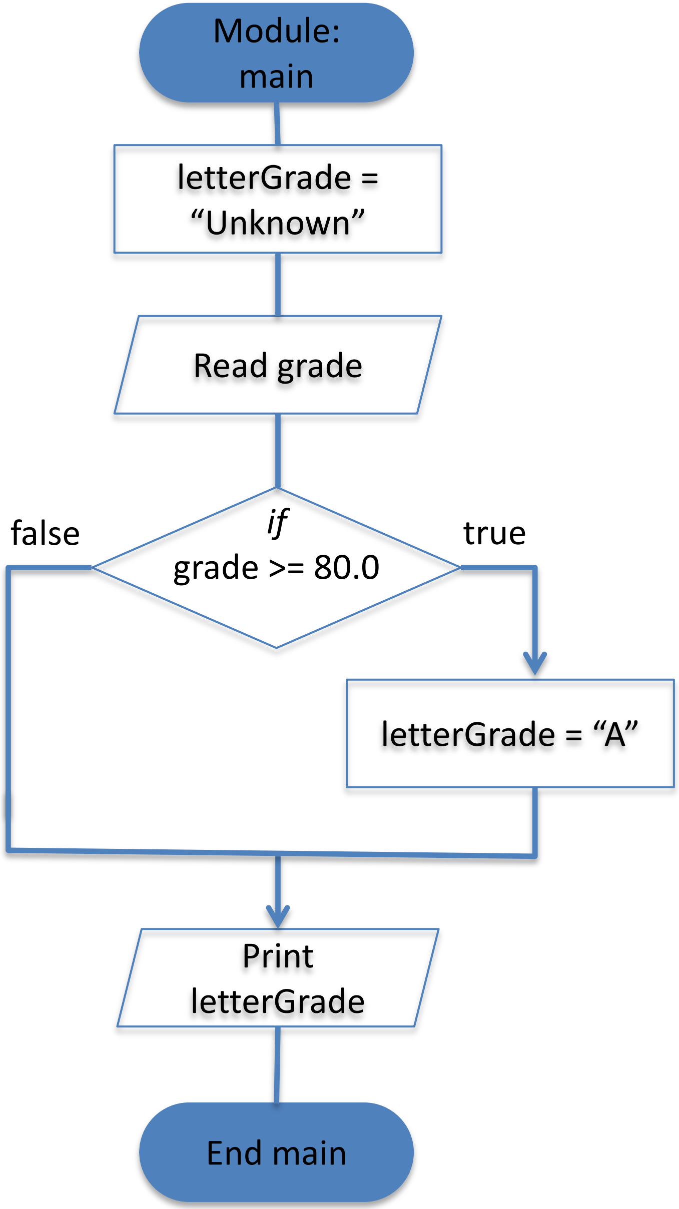
```
double grade;
String letterGrade;
Scanner input = new Scanner(System.in);

System.out.print("Enter % grade: ");
grade = input.nextDouble();

if (grade >= 80.0) {
    letterGrade = "A";
}

else {
    letterGrade = "Unknown";
}

System.out.println("Letter grade is: "
                  + letterGrade);
```

Path through program with test input: grade = 68.0



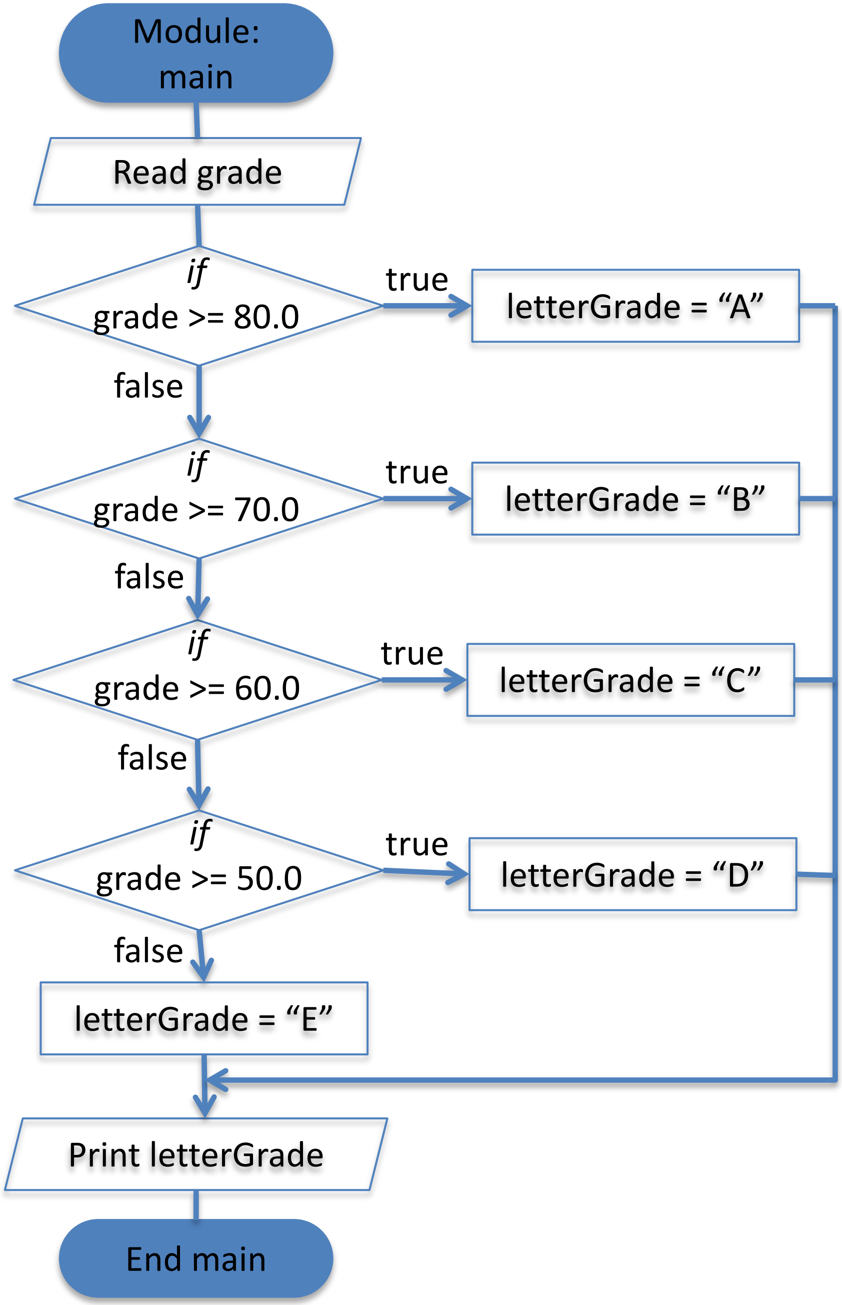
Example: if with no else

```
double grade;
String letterGrade = "Unknown";
Scanner input = new Scanner(System.in);

System.out.print("Enter % grade: ");
grade = input.nextDouble();

if (grade >= 80.0) {
     letterGrade = "A";
}

System.out.println("Letter grade is: "
                  + letterGrade);
```



More than 2 choices?
if … else if … else

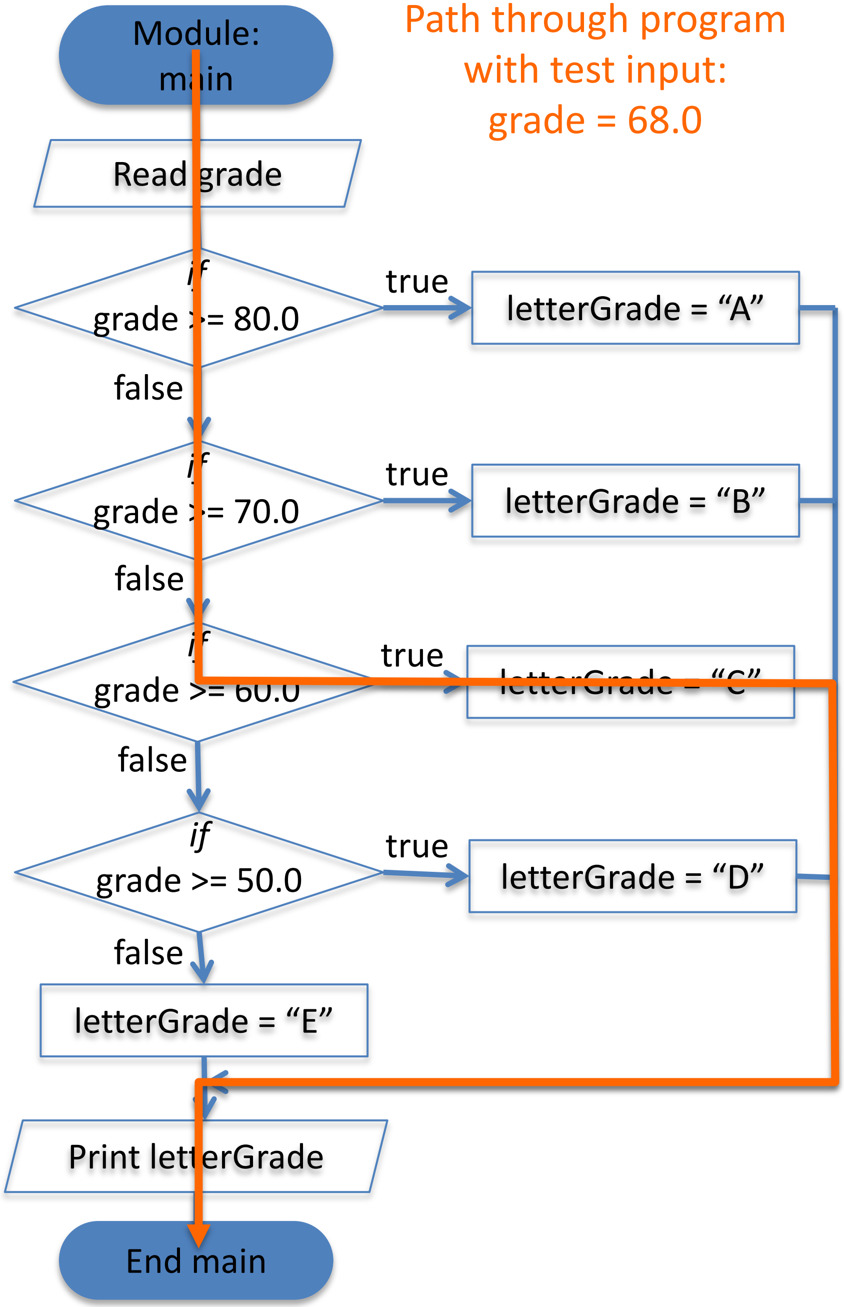
```
double grade;
String letterGrade;
Scanner input = new Scanner(System.in);

System.out.print(“Enter % grade: ”);
grade = input.nextDouble();

if (grade >= 80.0)
    letterGrade = “A”;
else if (grade >= 70.0)
    letterGrade = “B”;
else if (grade >= 60.0)
    letterGrade = “C”;
else if (grade >= 50.0)
    letterGrade = “D”;
else
    letterGrade = “E”;

System.out.println(“Letter grade is: ”
                  + letterGrade);
```



More than 2 choices?

if … else if … else

```
double grade;
String letterGrade;
Scanner input = new Scanner(System.in);

System.out.print("Enter % grade: ");
grade = input.nextDouble();

if (grade >= 80.0)
    letterGrade = "A";
else if (grade >= 70.0)
    letterGrade = "B";
else if (grade >= 60.0)
    letterGrade = "C";
else if (grade >= 50.0)
    letterGrade = "D";
else
    letterGrade = "E";

System.out.println("Letter grade is: "
                  + letterGrade);
```



Path through program with test input: grade = 68.0

Would it work to change the order in which conditions are checked?

```
double grade;
String letterGrade = “Unknown”;
Scanner input = new Scanner(System.in);

System.out.print(“Please enter the student’s grade: “);
grade = input.nextInt();

if (grade >= 50.0)
     letterGrade = “D”;
else if (grade >= 60.0)
     letterGrade = “C”;
else if (grade >= 70.0)
     letterGrade = “B”;
else if (grade >= 80.0)
     letterGrade = “A”;
else
     letterGrade = “E”;

System.out.println(“The student’s letter grade is: “ + letterGrade);
```

This program is incorrect.

It will set letterGrade to D for all marks above or equal to 50, and E for all marks below 50!

Switch statement

- An alternative to multi-way if-else statements, if the choice is based on the value of an integer or character expression (and in Java version 7 or later, Strings too)

```
Scanner kb =new Scanner(System.in);
String input = kb.next();
char letterGrade = input.charAt(0);

if (letterGrade == ‘A’)
  System.out.println(“Great”);
else if (letterGrade == ‘B’)
  System.out.println(“Good”);
else if (letterGrade == ‘C’)
  System.out.println(“Fair”);
else
  System.out.println(“Do better”);
```

```
Scanner kb =new Scanner(System.in);
String input = kb.next();
char letterGrade = input.charAt(0);

switch (letterGrade){
  case ‘A’:
    System.out.println(“Great”);
    break;
 case ‘B’:
    System.out.println(“Good”);
    break;
 case ‘C’:
    System.out.println(“Fair”);
    break;
 default:
    System.out.println(“Do
better”);
    break;
}
```

Switch statement – Example 2

Omitted is some code that sets *letterGrade*

```
char letterGrade;

if (letterGrade == ‘A’ ||
     letterGrade == ‘B’ ||
     letterGrade == ‘C’ ||
     letterGrade == ‘D’)
   System.out.println(“Pass”);
else
   System.out.println(“Fail”);
```

```
char letterGrade;

switch (letterGrade){
   case ‘A’:
   case ‘B’:
   case ‘C’:
   case ‘D’:
     System.out.println(“Pass”);
     break;
  default:
     System.out.println(“Fail”);
     break;
}
```

ASHESI

Switch statement – Example 3

Omitted is some code that sets *letterGrade*

```
char letterGrade;

if (letterGrade == ‘A’)
    System.out.println(“Great!”);

if (letterGrade == ‘A’ ||
    letterGrade == ‘B’ ||
    letterGrade == ‘C’ ||
    letterGrade == ‘D’)
  System.out.println(“Pass”);
else
  System.out.println(“Fail”);
```

```
char letterGrade;

switch (letterGrade){
  case ‘A’:
    System.out.println(“Great!”);
  case ‘B’:
  case ‘C’:
  case ‘D’:
    System.out.println(“Pass”);
    break;
 default:
    System.out.println(“Fail”);
    break;
}
```

Repeating a group of instructions many times:

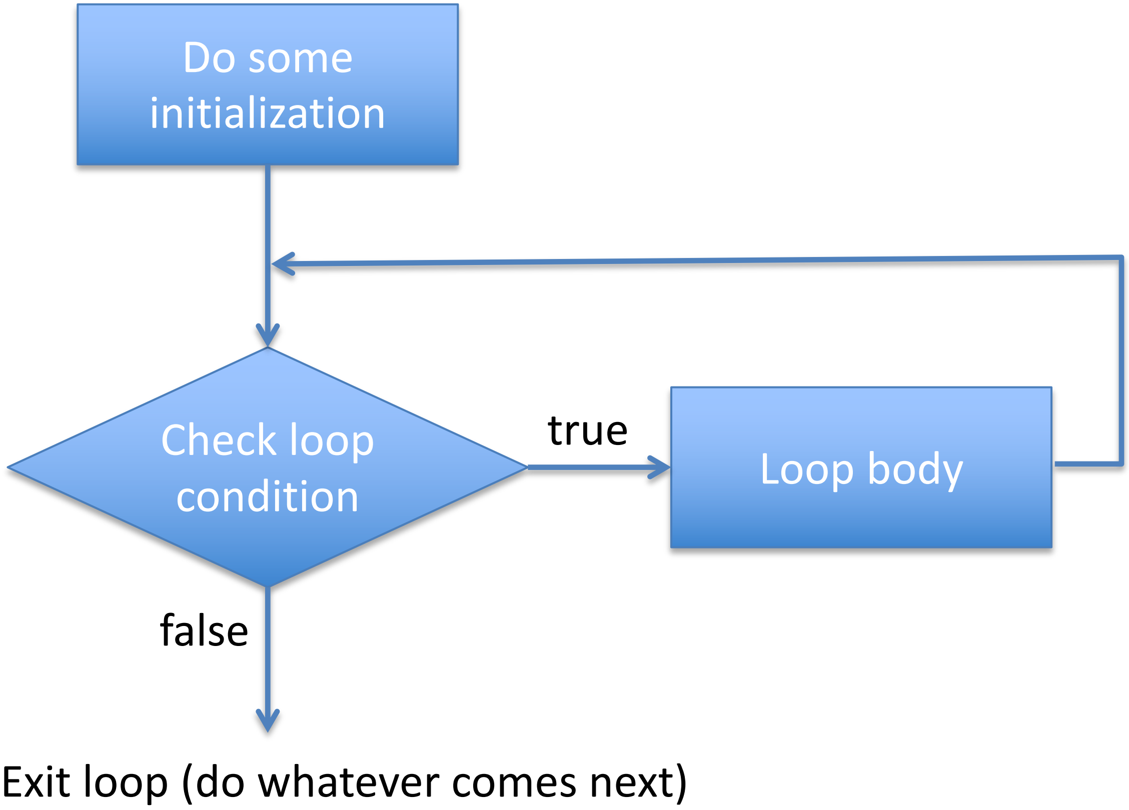
LOOPS

While loops

- Repeatedly executes a block of code as long as a boolean expression is true
- Stops executing the block of code once the boolean expression becomes false

```
while (boolean_expression) {

    // what to do while true

}
```

Flowchart showing how while-loops work



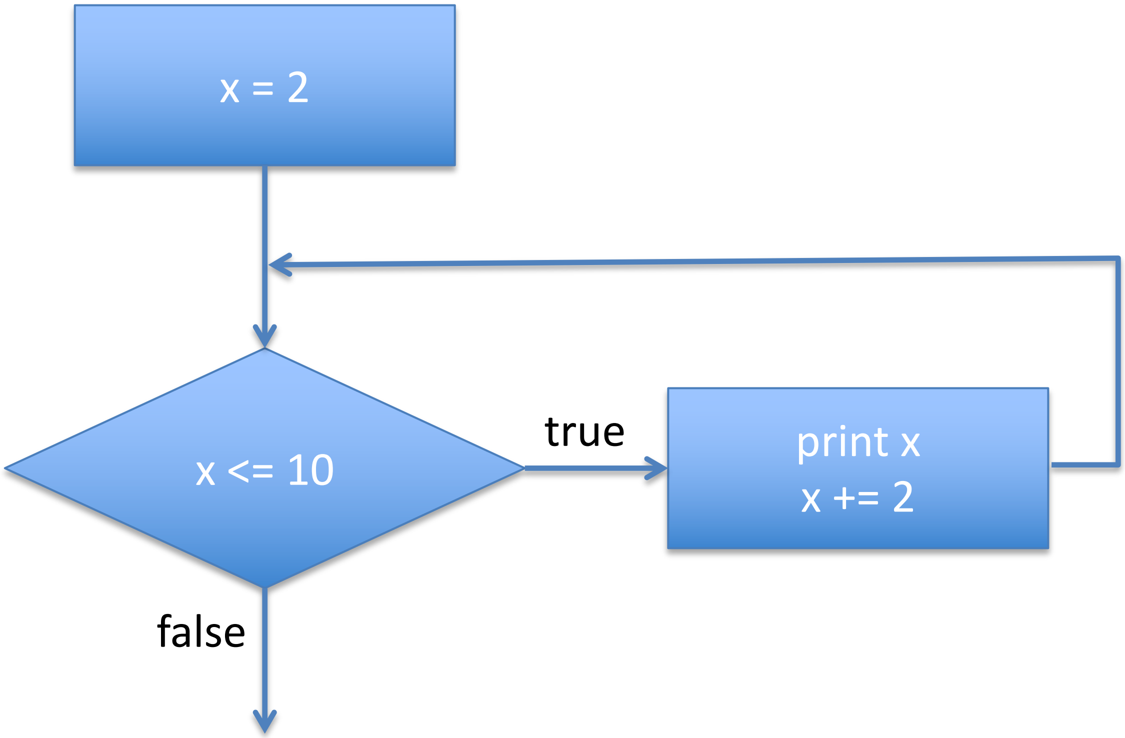
While loop example

- This code prints out even numbers up to 10
- The loop runs for 5 iterations before exiting

```
int x = 2;

while (x <= 10) {
    System.out.println(x);

    x += 2;
}
```

Flowchart for our code segment

```
int x = 2;

while (x <= 10) {
    System.out.println(x);

    x += 2;
}
```



x = 2

x <= 10

true

print x
x += 2

false

Exit loop (do whatever comes next)

While loop example

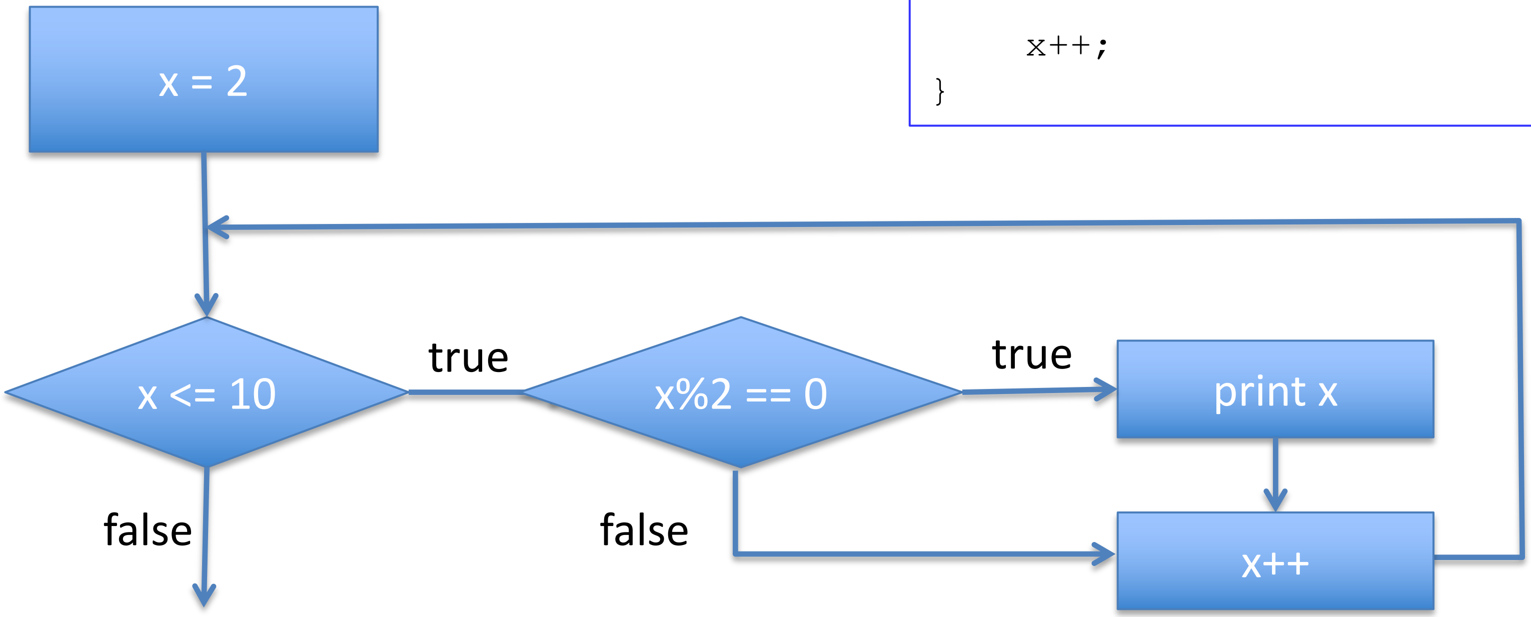
- This code also prints out even numbers up to 10
- The loop runs for 9 iterations before exiting

```
int x = 2;

while (x <= 10) {
    if (x%2 == 0)
        System.out.println(x);

    x++;
}
```

- For a more interesting example, see CountMultiples.java

Flowchart for 2nd code segment

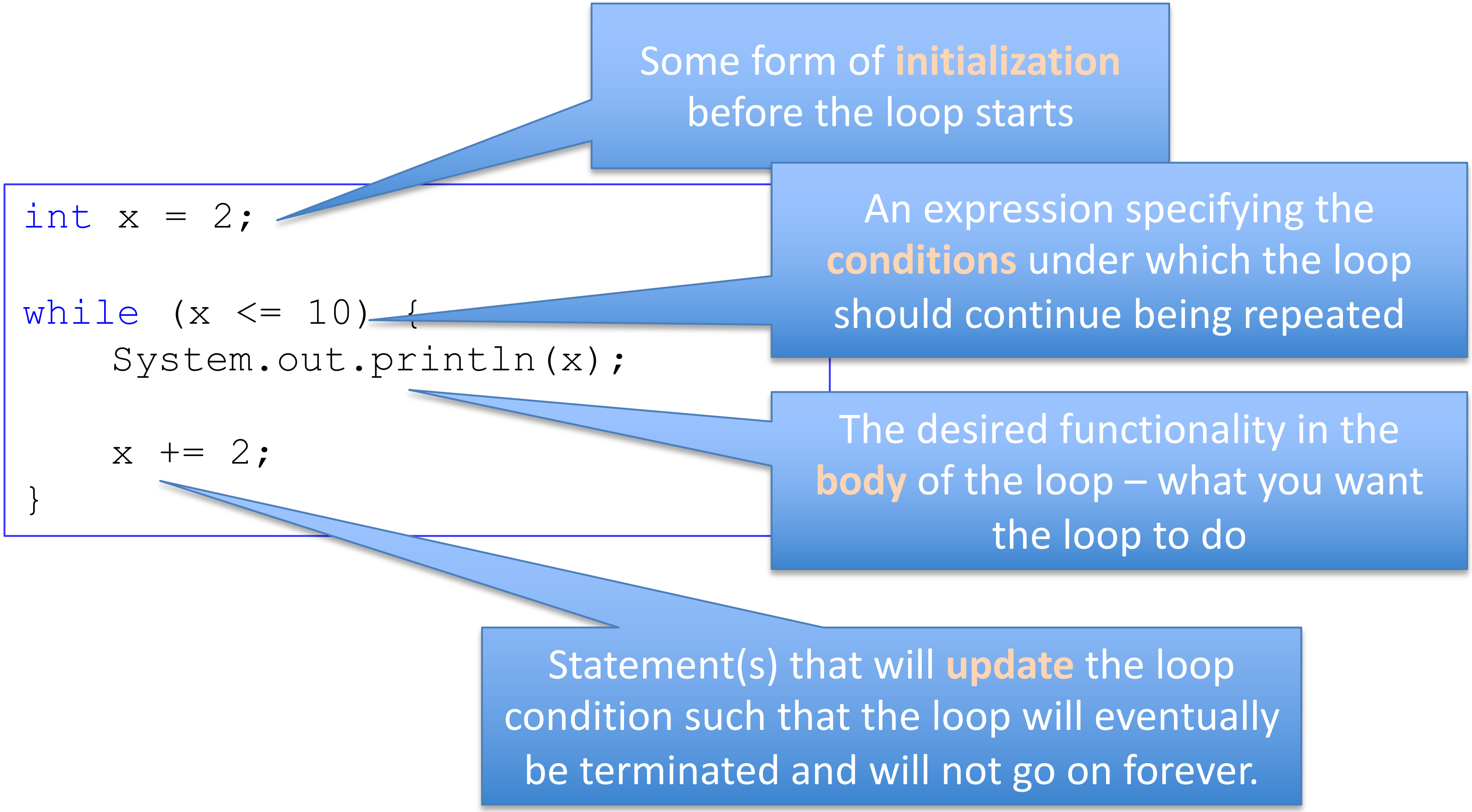
```
int x = 2;

while (x <= 10) {
     if (x%2 == 0)
          System.out.println(x);

     x++;
}
```



Exit loop (do whatever comes next)

Essential parts of a loop

- Make sure your loops always have these parts



```
int x = 2;

while (x <= 10) {
    System.out.println(x);

    x += 2;
}
```

Some form of **initialization** before the loop starts

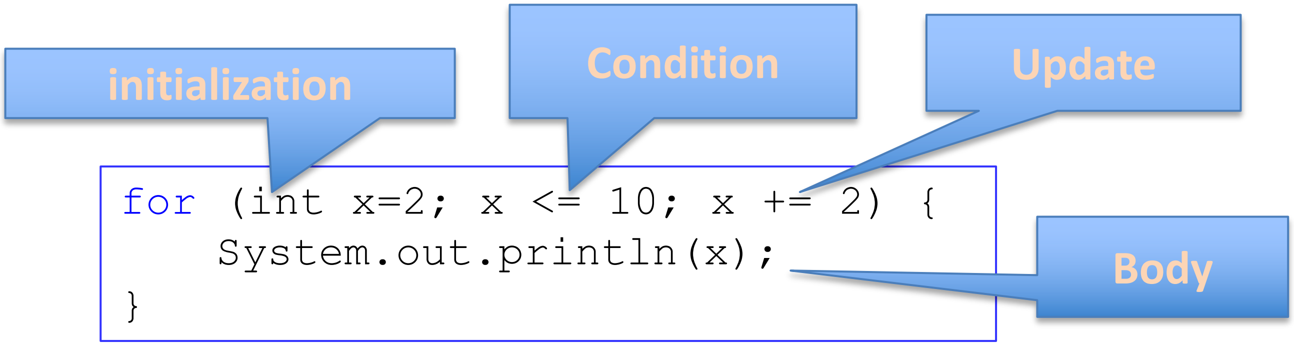
An expression specifying the **conditions** under which the loop should continue being repeated

The desired functionality in the **body** of the loop – what you want the loop to do

Statement(s) that will **update** the loop condition such that the loop will eventually be terminated and will not go on forever.

For loops

- A for-loop has the same elements as a while-loop, but structured differently
  - The header of a for-loop has 3 distinct parts, separated by semi-colons
- The for-loop below behaves exactly like the while-loop on the previous slide



initialization

Condition

Update

Body

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

- For another example, see NameTest.java

Recap: While loops & for-loops

- Repeatedly executes a block of code as long as a given condition (boolean expression) is true
- Stops executing the block of code once the condition becomes false
- Minimum number of times loop can run: 0 (if condition is false from the start)

```
while (condition) {

    // what to do while true

}
```

```
for (initialization; condition; update){

    // what to do while true

}
```

More about loops: *do … while*

- Repeatedly executes a block of code as long as a condition (boolean expression) is true
- Stops executing the block of code once the boolean expression becomes false
- Minimum number of times loop can run: 1

```
do {

    // what to do while true

} while (condition);
```

How many times will each of these loops execute?

```java
int x = 10;

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

```java
int x = 10;

do {
    System.out.println(x);
    x++;
} while (x < 10);
```

Recap: Boolean Expressions

- Flow of control (branching, looping) depends on conditions or boolean expressions

- What is a boolean expression?
  - A boolean value or variable
    - E.g. boolean w = false
  - An expression that returns a boolean value, E.g.
    - Greater than, e.g.: x > 0
    - Greater than or equal to, e.g.: x >= 0
    - Less than, e.g.: x < 0
    - Less than or equal to, e.g.: x <= 0
    - Equals (primitive types) , e.g.: y == 4
    - Equals (Strings) , e.g.: name.equals("ayorkor")

- See: SimpleConditions.java,

ASHESI

What is a Boolean Expression? continued

- Two boolean expressions connected with boolean operators
  - AND: &&
    - E.g. (x >= 0 && x < 10)
  - OR: ||
    - E.g. (x < 0 || x >= 10)

ASHESI

Remember our truth tables

x	*y*	*x || y*
true	true	true
true	false	true
false	true	true
false	false	false

Notice that if x is **true**, x || y is **true**, regardless of the value of y

x	*y*	*x && y*
true	true	true
true	false	false
false	true	false
false	false	false

Notice that if x is **false**, x && y is **false**, regardless of the value of y

ASHESI

What is a Boolean Expression? continued

- The negation of a boolean expression, e.g.
  - Not equals (primitive types) , e.g.: z != 4
  - Not equals (Strings) , e.g.: !name.equals(“ayorkor”)
  - !(x > 0 && x <= 10)
    - Note that this is is the same as (x <= 0 || x > 10)
    - The latter way is a less confusing way of writing the expression
- See: PersonDescription.java

ASHESI

Nested blocks

- You can have an if-else block inside an if-block or else block
  - See PersonDescription2.java
- You can also have loops within if/else blocks, if/else blocks within loops, loops within loops etc
- The possibilities are endless – it all depends on the required program logic

ASHESI

Nested Loops

- Some problems require the use of nested loops – loops within loops
- Consider the problem of printing multiplication tables. See:
  - MultiplicationTable1.java (uses nested while loops)
  - MultiplicationTable2.java (uses nested for loops)

Errors & Debugging: Off-by-one errors

- An off-by-one error happens when you execute a loop one fewer or one less time than you mean to do.
- The following code which tries to implement multiplication by repeated addition, has an off-by-one error. Can you find it?

```
public class Multiply {
    public static void main(String[] args) {
        int x, y, result;
        Scanner input = new Scanner(System.in);

        System.out.println(“Input positive integers x and y: “);
        x = input.nextInt();
        y = input.nextInt();

        result = 0;
        for (int i=1; i<y; i++){
            result += x;
        }
    }
}
```

Off-by-one errors

- How did we identify the off-by-one error in the previous code?
  - By thinking about it analytically
    - (starting the counter at 1 and going as long as the counter is less than y results in executing the loop y-1 times when we really want to do it y times)
  - By trying with an example
    - (if we step through the code with x=2 and y=3, result ends up being 4 instead of 6)
  - By putting print out statements in the loop to print what happens each time

Additional Example Programs

- See: UnsecureLogin1.java
  - Illustrates the use of the String methods `equals()` and `equalsIgnoreCase()` to check a username and password
- See: UnsecureLogin2.java
  - A modified version of UnsecureLogin1.java that makes use of nested if-statements

Another example program using loops

- Suppose I wanted the user to be able to enter some text at the keyboard and I'll count how many words are in their input
  - First version: CountWords.java
    - This version is a good start, but doesn't work well if the user enters multiple spaces between their words
  - Second version: CountWords2.java
    - This version is better because it is able to handle when the user enters multiple spaces between their words. But it still has trouble if the user starts their sentence with a space
  - Third version: CountWords3.java
    - This version correctly handles if the input starts with spaces

With loops, we can potentially do some dangerous things!

- What happens when I run this block of code?

```java
int x = 2;

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

- How about this one?

```java
int x = 2;

while (x > 10) {
    System.out.println(x);
    x++;
}
```

More Example Programs Using Loops

- Here is a program that asks the user for input and then tests for incorrect input and repeatedly prompt the user until they enter good input.
  - See CountMultiples2.java

- Also, note that instead of the boolean expression (goodInput == false) in the above program, we could use (!goodInput)
  - See CountMultiples3.java

Short-cut evaluation of boolean expressions

- For OR expressions: *Expression_A || Expression_B*
  - If *Expression_A* is **true**, *Expression_B* is never evaluated
  - Example:
    - (x < 0 || x > 10)
    - If x is -3, the expression evaluates to **true** without evaluating x > 10

- For AND expressions: *Expression_A && Expression_B*
  - If *Expression_A* is **false**, *Expression_B* is never evaluated
  - Example:
    - (x > 0 && x < 10)
    - If x is -3, the expressions evaluates to **false** without evaluating x < 10

ASHESI

Additional Concepts

- In your reading of the textbook, look out for:
  - The conditional / ternary operator (Sect. 3.1)
    - E.g. `int abs = (x < 0) ? -x : x`
  - The System.exit() method (Sect 3.1)
  - Enumerations (Sect 3.3)
  - For-Each statement (used with enumerations) (Sect 4.1)
  - The continue and break keywords, used in loops (Sect 4.2)
  - The assert keyword (Sect 4.2)

- Also pay particular attention to the boxes labeled "Gotcha!"

ASHESI