Chapter Five: Conditionals and Loops

CTEC 150, Fall 2019

Jay Jain

Bossier Parish Community College

jjain@bpcc.edu

August 22, 2019

Overview

- **5.1: Boolean Expressions**
- 5.2: The `if` Statement
- 5.3: Comparing Data
- 5.4: The `while` Statement
- 5.5: Iterators
- 5.6: The `ArrayList` Class

5.1 Boolean Expressions

- The flow of control is the order in which statements are executed

5.1 Boolean Expressions

- The flow of control is the order in which statements are executed
- All Java programs start from the `main` method

5.1 Boolean Expressions

- The flow of control is the order in which statements are executed
- All Java programs start from the `main` method
- A conditional statement also known as a selection statement allows us to choose which statement will be executed next

5.1 Boolean Expressions

- The flow of control is the order in which statements are executed
- All Java programs start from the `main` method
- A conditional statement also known as a selection statement allows us to choose which statement will be executed next
- Each decision we make is based on a boolean expression that evaluates to either `true` or `false`

5.1 Boolean Expressions

- The flow of control is the order in which statements are executed
- All Java programs start from the `main` method
- A conditional statement also known as a selection statement allows us to choose which statement will be executed next
- Each decision we make is based on a boolean expression that evaluates to either `true` or `false`

```
if (happiness <= 0){
    System.out.println("I'm sad face.");
}else if(happiness > 0 && happiness < 100){
    System.out.println("Somewhere in between");
}else{
    System.out.println("I'm ecstatic!!!!!");
}
```

Boolean Expressions

Equality or relational operators return boolean results (true or false)

== equal to
!= not equal to
< less than
> greater than
<= less than or equal to
<= greater than or equal to

Reminder

Note the difference between the equality operator (==) and the assignment operator(=)

Logical Operators

- Logical NOT is a unary operator
- Logical AND and OR are binary operators

!	Logical NOT
&&	Logical AND
||	Logical OR

a	!a
true	false
false	true

Truth Table

- A truth table shows all possible true-false combinations
- There are four possible combinations for two variables

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

5.2: The `if` Statement

- 5.1: Boolean Expressions
- **5.2: The `if` Statement**
- 5.3: Comparing Data
- 5.4: The `while` Statement
- 5.5: Iterators
- 5.6: The `ArrayList` Class

The if Statement

Structure of the if Statement

```
if( condition ){
    statement;
}
```

The `if` Statement

Structure of the if Statement

```
if( condition ){
    statement;
}
```

- `if` is a reserved word in Java

The `if` Statement

Structure of the if Statement

```
if( condition ){
    statement;
}
```

- `if` is a reserved word in Java
- The condition must be a boolean expression

The `if` Statement

Structure of the if Statement

```
if( condition ){
    statement;
}
```

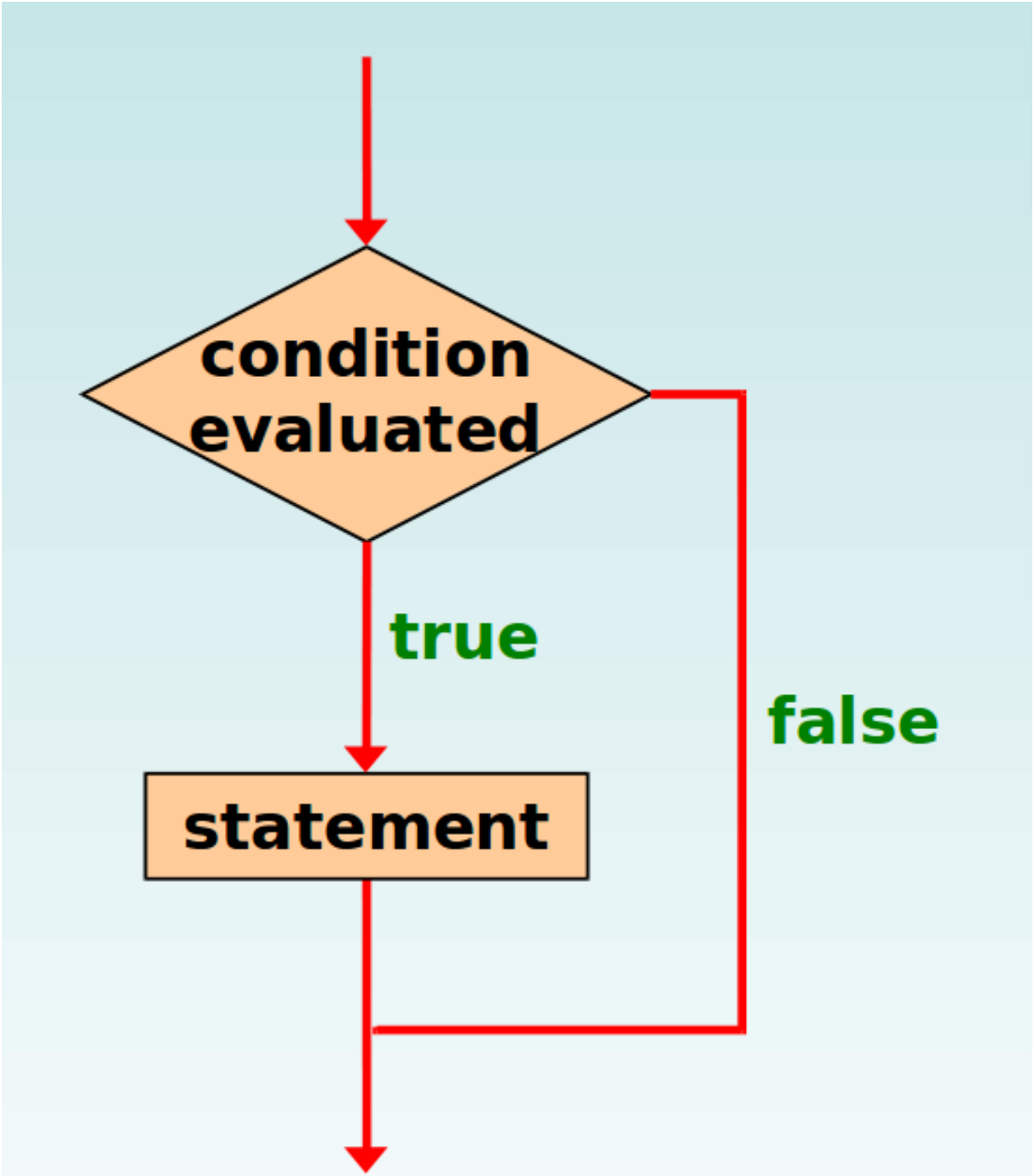
- `if` is a reserved word in Java
- The condition must be a boolean expression
- If the condition is true, then the statement is executed

The `if` Statement

Structure of the if Statement

```
if( condition ){
    statement;
}
```

- `if` is a reserved word in Java
- The condition must be a boolean expression
- If the condition is true, then the statement is executed
- If the condition is false, the statement is skipped


condition
evaluated
true
false
statement

Review

1

```
int total = 34;
int stock = 24;
int warehouse = 8;
if( total != stock + warehouse ){
    inventoryError = true;
}
```

Review

1

```
int total = 34;
int stock = 24;
int warehouse = 8;
if( total != stock + warehouse ){
    inventoryError = true;
}
```

2

```
boolean found = false;
boolean done = true;
if( found || !done ){
    inventoryError = true;
}
```

The `if - else` Statement

Structure of the if-else Statement

```
if( condition ){
    statement1;
}else{
    statement2;
```

The `if - else` Statement

Structure of the if-else Statement

```
if( condition ){
    statement1;
}else{
    statement2;
```

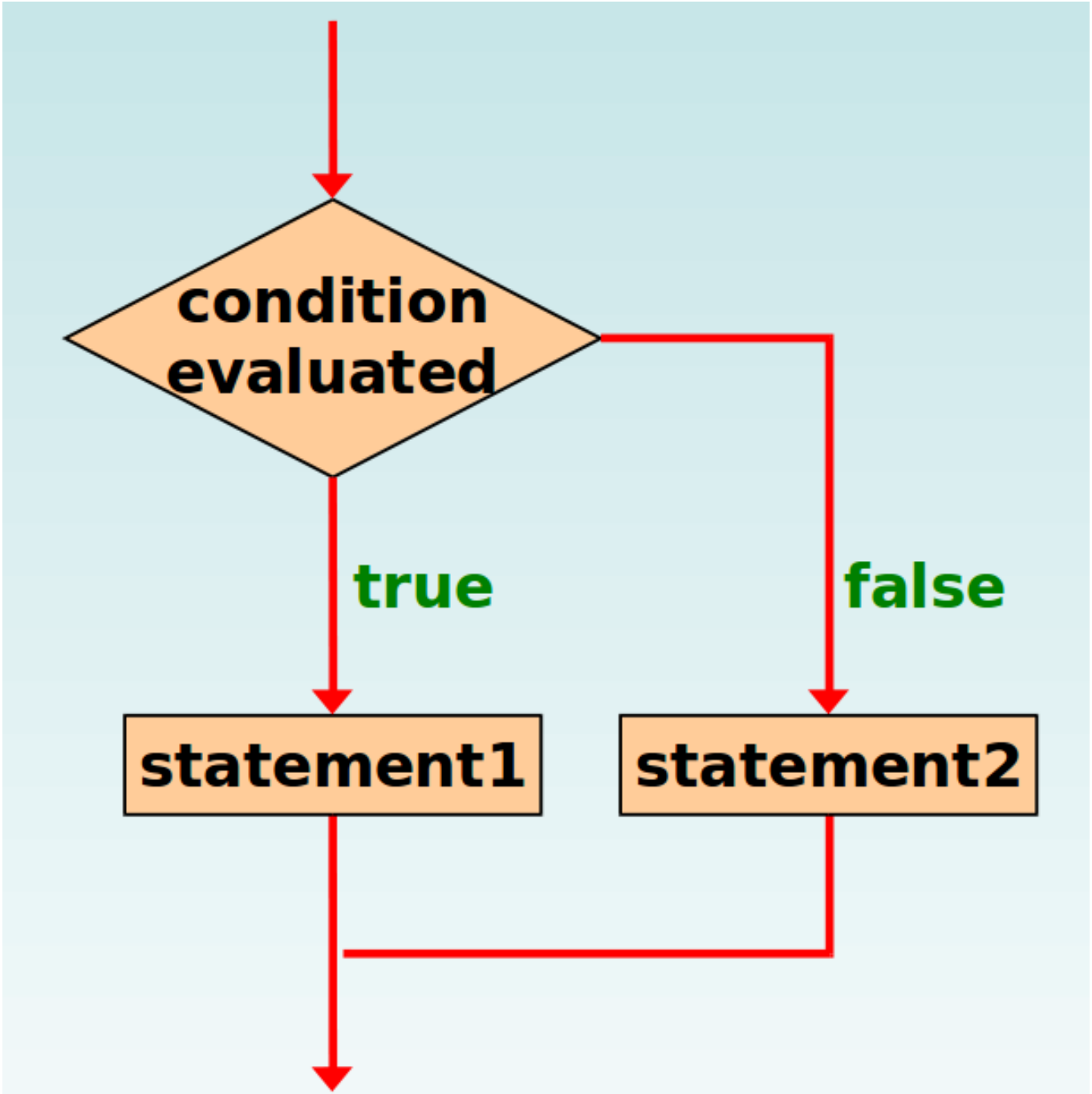
- If the condition is true, statement1 is executed

The `if - else` Statement

Structure of the if-else Statement

```
if( condition ){
    statement1;
}else{
    statement2;
```

- If the condition is true, statement1 is executed
- If the condition is false, statement2 is executed


condition
evaluated
true
false
statement1
statement2

Indentation

Warning - Bad Practice

```
if(depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
    delta = 0;
```

Indentation

Warning - Bad Practice

```
if(depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
    delta = 0;
```

- Indentation is for the human reader and is ignored by the compiler

Indentation

Warning - Bad Practice

```
if(depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
    delta = 0;
```

- Indentation is for the human reader and is ignored by the compiler
- The variable `delta` will be set to 0 no matter what in this case

Indentation

Warning - Bad Practice

```
if(depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Reseting Delta");
    delta = 0;
```

- Indentation is for the human reader and is ignored by the compiler
- The variable `delta` will be set to 0 no matter what in this case

Block Statements

```
if(depth >= UPPER_LIMIT){
    delta = 100;
}else{
    System.out.println("Reseting Delta");
    delta = 0;
}
```

5.3: Comparing Data

- 5.1: Boolean Expressions
- 5.2: The `if` Statement
- **5.3: Comparing Data**
- 5.4: The `while` Statement
- 5.5: Iterators
- 5.6: The `ArrayList` Class

Comparing Float Values

- Two floating point values are equal only if their underlying binary representations match exactly

Comparing Float Values

- Two floating point values are equal only if their underlying binary representations match exactly
- Arithmetic operations result in slight differences over time

Comparing Float Values

- Two floating point values are equal only if their underlying binary representations match exactly
- Arithmetic operations result in slight differences over time
- To determine equality of two floats:

```
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

Comparing Float Values

- Two floating point values are equal only if their underlying binary representations match exactly
- Arithmetic operations result in slight differences over time
- To determine equality of two floats:

```
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

- Tolerance could be set to 0.000001

Comparing Characters

- Java character data is based on Unicode character set

Comparing Characters

- Java character data is based on Unicode character set
- There is a numeric value for each character, and therefore an ordering

Comparing Characters

- Java character data is based on Unicode character set
- There is a numeric value for each character, and therefore an ordering

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Strings

- Cannot use relational operator (==) to compare `String`

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`
- returns 0 if name1 and name2 are equal

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`
- returns 0 if name1 and name2 are equal
- returns negative value if name1 is less than name2

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`
- returns 0 if name1 and name2 are equal
- returns negative value if name1 is less than name2
- returns positive value if name1 is greater than name2

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`
- returns 0 if name1 and name2 are equal
- returns negative value if name1 is less than name2
- returns positive value if name1 is greater than name2
- This is based on the order of characters or lexicographic ordering

Comparing Strings

- Cannot use relational operator (==) to compare `String`
- The `String` class contains the `compareTo` method to compare `Strings`
- `name1.compareTo(name2)`
- returns 0 if name1 and name2 are equal
- returns negative value if name1 is less than name2
- returns positive value if name1 is greater than name2
- This is based on the order of characters or lexicographic ordering
- The string "Great" comes before the string "fantastic" because all of the uppercase letters come before all of the lowercase letters in Unicode

5.4: The `while` Statement

- 5.1: Boolean Expressions
- 5.2: The `if` Statement
- 5.3: Comparing Data
- **5.4: The `while` Statement**
- 5.5: Iterators
- 5.6: The `ArrayList` Class

The while Statement

Structure of the while Statement

```
while ( condition ){
    statement1;
}
```

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

- If condition is true, statement is executed

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

- If condition is true, statement is executed
- Condition is evaluated again, and if it is still true, statement is executed again

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

- If condition is true, statement is executed
- Condition is evaluated again, and if it is still true, statement is executed again
- Statement is executed repeatedly until condition becomes false

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

- If condition is true, statement is executed
- Condition is evaluated again, and if it is still true, statement is executed again
- Statement is executed repeatedly until condition becomes false
- Repetition statements allow us to execute a statement multiple times

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

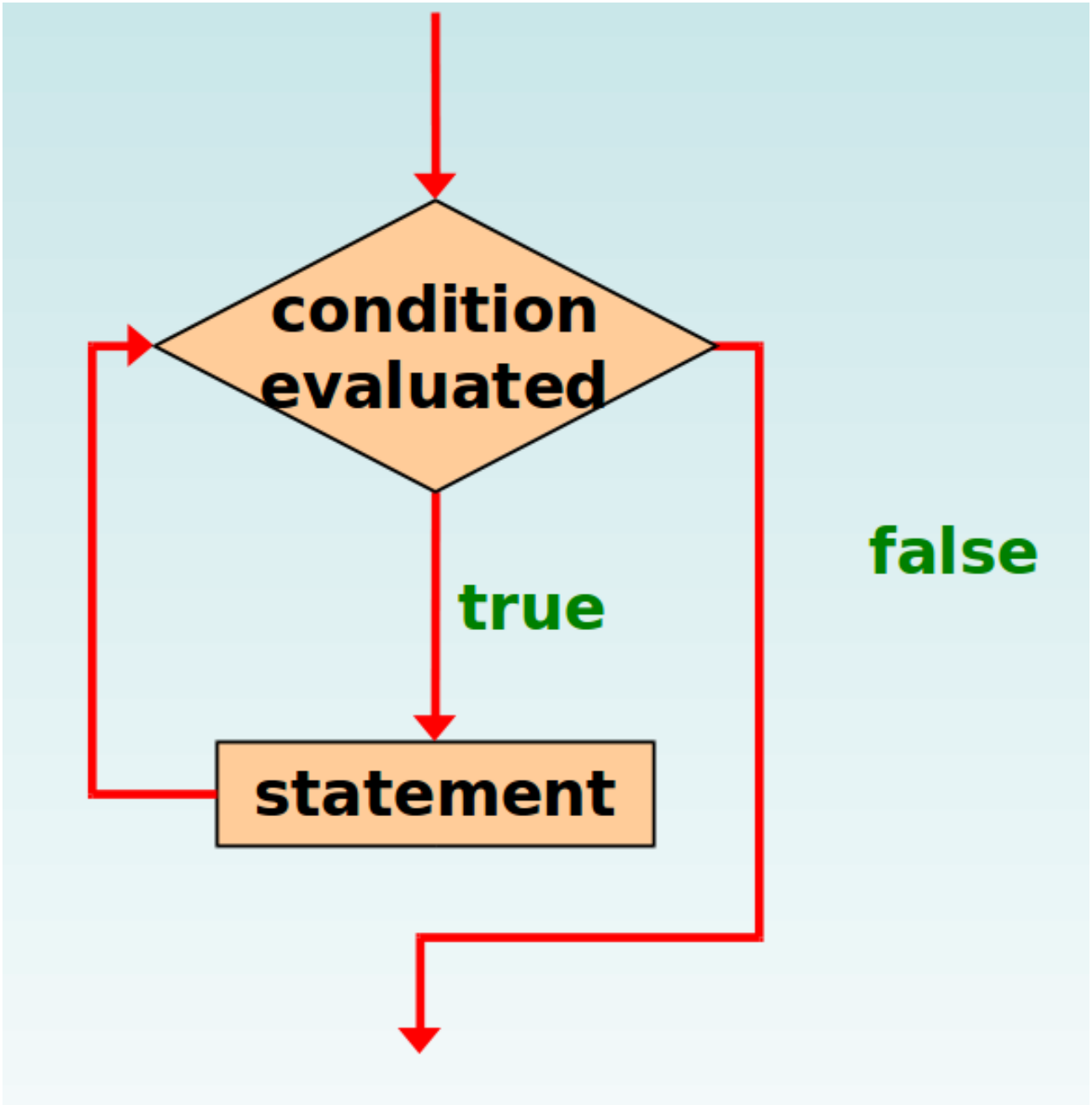
- If condition is true, statement is executed
- Condition is evaluated again, and if it is still true, statement is executed again
- Statement is executed repeatedly until condition becomes false
- Repetition statements allow us to execute a statement multiple times
- Referred to as loops

The `while` Statement

Structure of the `while` Statement

```
while ( condition ){
    statement1;
}
```

- If condition is true, statement is executed
- Condition is evaluated again, and if it is still true, statement is executed again
- Statement is executed repeatedly until condition becomes false
- Repetition statements allow us to execute a statement multiple times
- Referred to as loops
- Controlled by boolean expressions


condition
evaluated
true
false
statement

Example of `while` loop

Structure of the `while` Statement

```
int count = 1;
while (count <= 5){
    System.out.println(count);
    count++;
}
```

Example of `while` loop

Structure of the `while` Statement

```
int count = 1;
while (count <= 5){
    System.out.println(count);
    count++;
}
```

- If the condition is false initially, the statement never gets executed

Example of `while` loop

Structure of the `while` Statement

```
int count = 1;
while (count <= 5){
    System.out.println(count);
    count++;
}
```

- If the condition is false initially, the statement never gets executed
- The body of a `while` loop will execute zero or more times

Infinite Loops

- The body of a while loop eventually must make the condition false

Infinite Loops

- The body of a while loop eventually must make the condition false
- If not, it is called an *infinite loop*

Infinite Loops

- The body of a while loop eventually must make the condition false
- If not, it is called an *infinite loop*
- Always double check your logic to ensure that the loop will terminate and will not be infinite

Infinite Loops

- The body of a while loop eventually must make the condition false
- If not, it is called an *infinite loop*
- Always double check your logic to ensure that the loop will terminate and will not be infinite

Infinite Loop Example

```
int count = 1;
while (count <= 25){
    System.out.println(count);
    count= count - 1;
}
```

Nested Loop Example

How many times will "Here" be printed?

```
count1 = 1;
while (count1 <= 10){
    count2 = 1;
    while (count2 < 20){
     System.out.println("Here");
     count2++;
    }
    count1++;
}
```

Nested Loop Example

How many times will "Here" be printed?

```
count1 = 1;
while (count1 <= 10){
     count2 = 1;
     while (count2 < 20){
      System.out.println("Here");
      count2++;
     }
     count1++;
}
```

10 * 19 = 190 times

5.5: Iterators

- 5.1: Boolean Expressions
- 5.2: The `if` Statement
- 5.3: Comparing Data
- 5.4: The `while` Statement
- **5.5: Iterators**
- 5.6: The `ArrayList` Class

Iterators

- An iterator is an object that allows you to process a collection of items one at a time

Iterators

- An iterator is an object that allows you to process a collection of items one at a time
- Lets you step through each item in turn

Iterators

- An iterator is an object that allows you to process a collection of items one at a time
- Lets you step through each item in turn
- An iterator has an `hasNext` method that returns true if there is at least one more item to process
- The `next` method returns the next item

Iterators

- An iterator is an object that allows you to process a collection of items one at a time
- Lets you step through each item in turn
- An iterator has an `hasNext` method that returns true if there is at least one more item to process
- The `next` method returns the next item
- `Scanner` class is an iterator

Iterators

- An iterator is an object that allows you to process a collection of items one at a time
- Lets you step through each item in turn
- An iterator has an `hasNext` method that returns true if there is at least one more item to process
- The `next` method returns the next item
- `Scanner` class is an iterator
- Particularly useful when reading input from a file (URLDissector.java)

5.6: The `ArrayList` Class

- 5.1: Boolean Expressions
- 5.2: The `if` Statement
- 5.3: Comparing Data
- 5.4: The `while` Statement
- 5.5: Iterators
- **5.6: The `ArrayList` Class**

The ArrayList Class

- Stores a list of objects

Methods

```
boolean add(E obj)
void add(int index, E obj)
Object remove(int index)
Object get(int index)
boolean isEmpty()
int size()
```

The ArrayList Class

- Stores a list of objects
- Part of the `java.util` package

Methods

```
boolean add(E obj)
void add(int index, E obj)
Object remove(int index)
Object get(int index)
boolean isEmpty()
int size()
```

The ArrayList Class

- Stores a list of objects
- Part of the `java.util` package
- You can reference each object in list using a numeric index starting at 0 (not 1)

Methods

```
boolean add(E obj)
void add(int index, E obj)
Object remove(int index)
Object get(int index)
boolean isEmpty()
int size()
```

The ArrayList Class

- Stores a list of objects
- Part of the `java.util` package
- You can reference each object in list using a numeric index starting at 0 (not 1)
- Grows and shrinks as needed; adjusting capacity as necessary

Methods

```
boolean add(E obj)
void add(int index, E obj)
Object remove(int index)
Object get(int index)
boolean isEmpty()
int size()
```

The ArrayList Class

- Stores object of any class, but cannot store primitive data types (`int`,`float`,`double`)

Creating ArrayList

```
ArrayList<String> names = new ArrayList<String>();
ArrayList<Book> list = new ArrayList<Book>();
```

QUESTIONS ???