After Midterm Summary

Chapter (6-7-8)

Chapter 6 – Iteration

- **While statement: executes a block of code repeatedly.**
- **while Loop Examples**

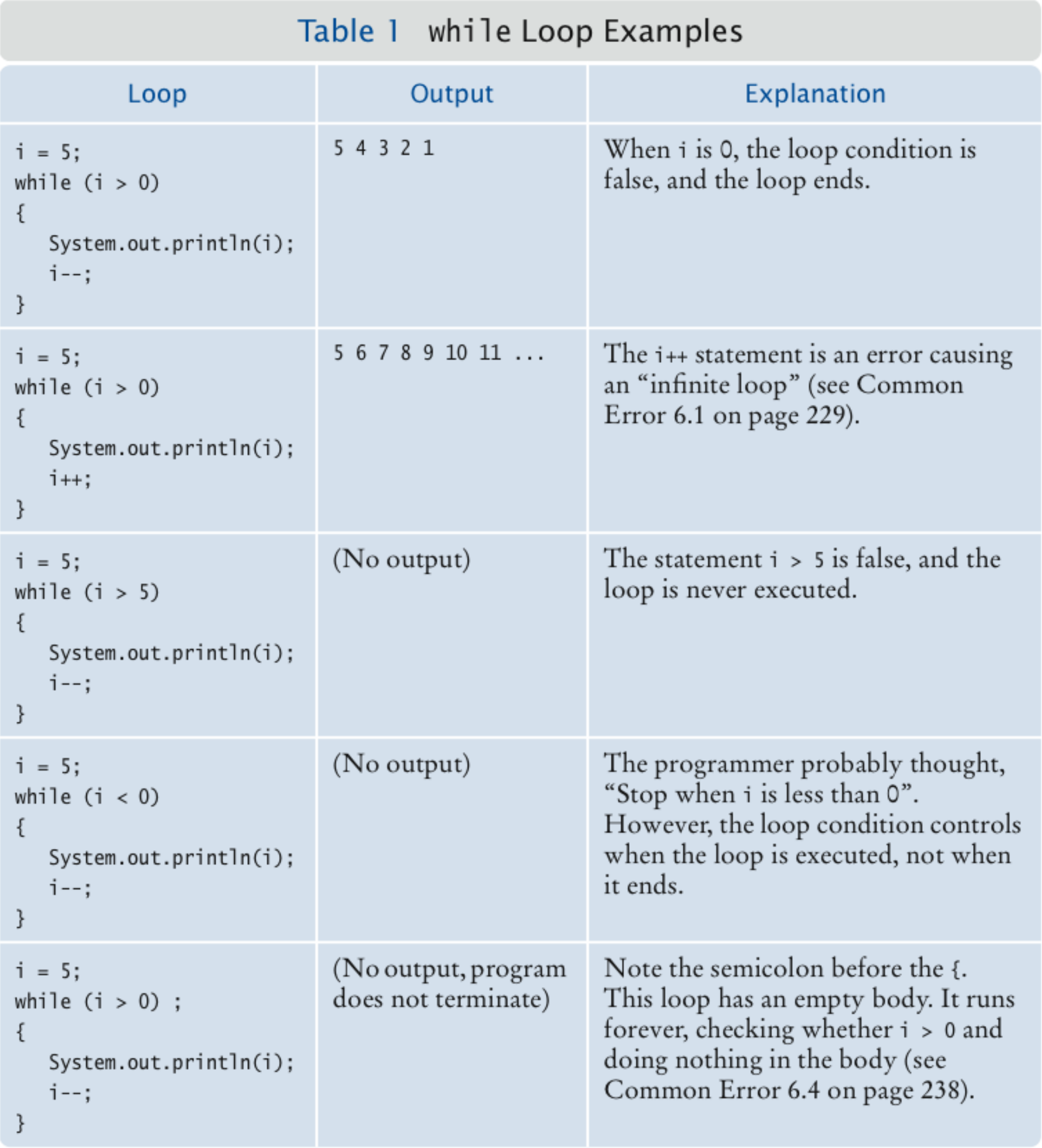


Table 1 while Loop Examples

Loop	Output	Explanation

`i = 5;`
`while (i > 0)`
`{`
`   System.out.println(i);`
`   i--;`
`}`
	5 4 3 2 1	When i is 0, the loop condition is false, and the loop ends.

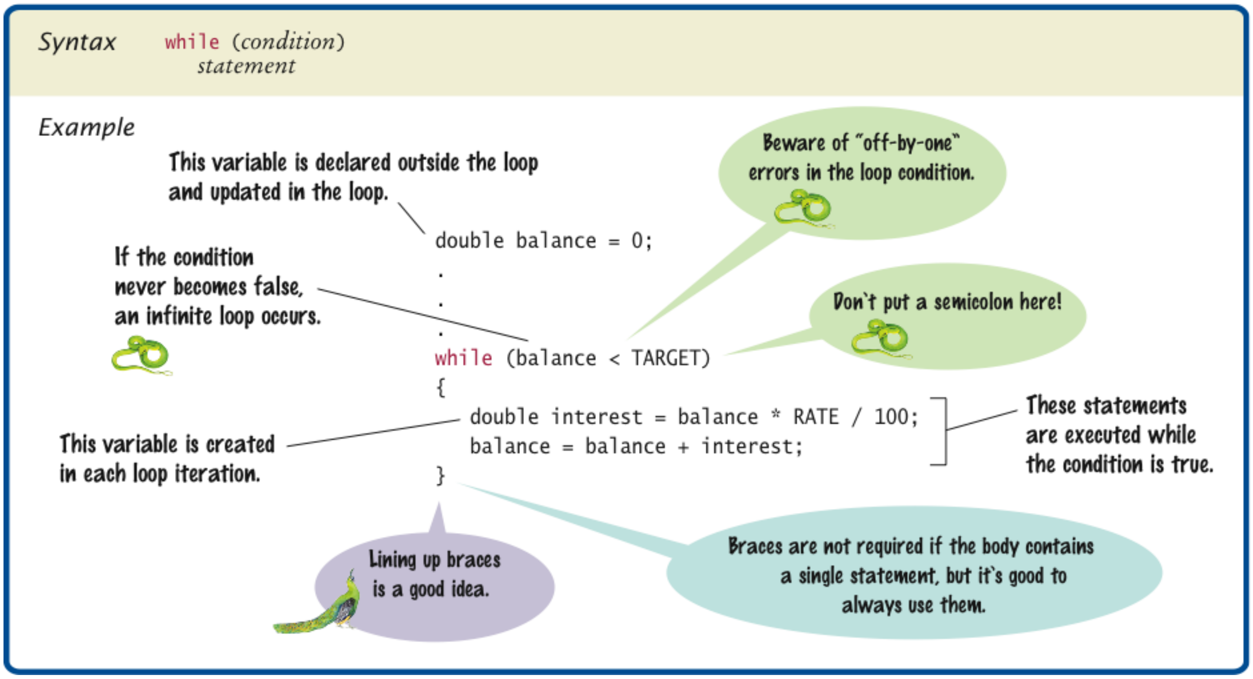
`i = 5;`
`while (i > 0)`
`{`
`   System.out.println(i);`
`   i++;`
`}`
	5 6 7 8 9 10 11 ...	The i++ statement is an error causing an "infinite loop" (see Common Error 6.1 on page 229).

`i = 5;`
`while (i > 5)`
`{`
`   System.out.println(i);`
`   i--;`
`}`
	(No output)	The statement i > 5 is false, and the loop is never executed.

`i = 5;`
`while (i < 0)`
`{`
`   System.out.println(i);`
`   i--;`
`}`
	(No output)	The programmer probably thought, "Stop when i is less than 0". However, the loop condition controls when the loop is executed, not when it ends.

`i = 5;`
`while (i > 0) ;`
`{`
`   System.out.println(i);`
`   i--;`
`}`
	(No output, program does not terminate)	Note the semicolon before the {. This loop has an empty body. It runs forever, checking whether i > 0 and doing nothing in the body (see Common Error 6.4 on page 238).

Syntax 6.1 The `while` Statement



Syntax `while` (*condition*)
statement

Example

```
double balance = 0;
.
.
.
while (balance < TARGET)
{
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

This variable is declared outside the loop and updated in the loop.

If the condition never becomes false, an infinite loop occurs.

This variable is created in each loop iteration.

Beware of "off-by-one" errors in the loop condition.

Don't put a semicolon here!

These statements are executed while the condition is true.

Lining up braces is a good idea.

Braces are not required if the body contains a single statement, but it's good to always use them.

Self Check 6.1

How often is the following statement in the loop executed?

while (false) statement;

Answer: Never

Infinite Loops

- Example:

```
int years = 0;
while (years < 20)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

- Loop runs forever — must kill program

Q. What is an infinite loop and how can you terminate a program that executes an infinite loop?

Answer: An infinite loop is a loop that will keep executing and never terminates.
It causes a run-time error where the program gets stuck looping.
The way to terminate the infinite loop is to **kill the process**.

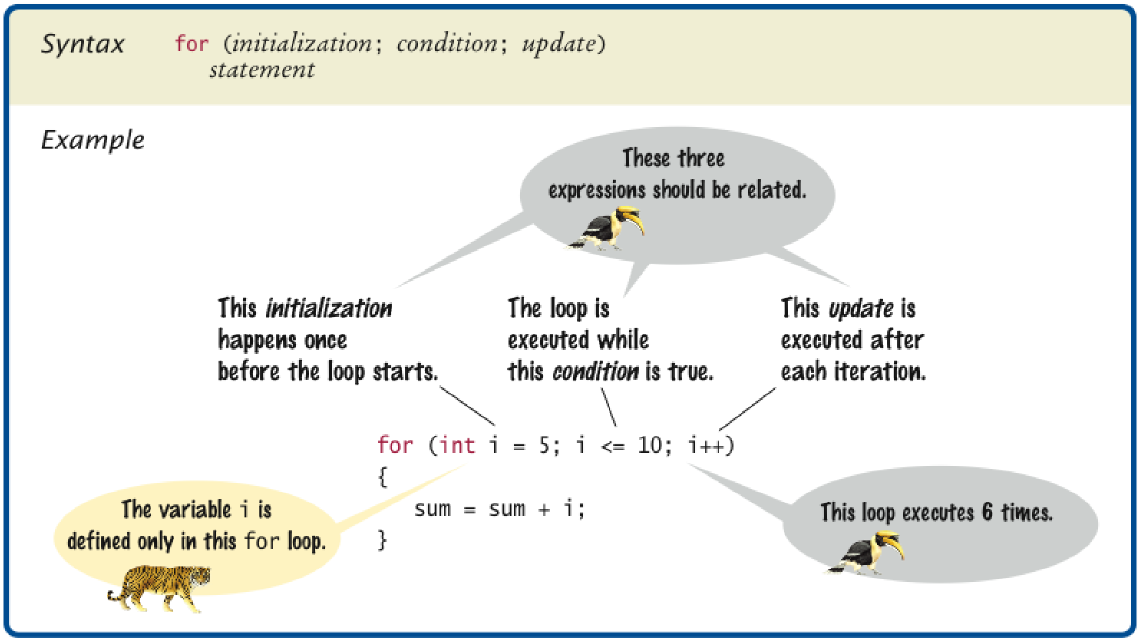
- **Off-by-one error: a loop executes one too few, or one too many, times**

for Loops

- Example:

```
for (int i = 1; i <= n; i++)
   {
      double interest = balance * rate / 100;
      balance = balance + interest;
   }
```

Syntax 6.2 The `for` Statement



Syntax `for (`*initialization*`;` *condition*`;` *update*`)`
statement

Example

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

These three expressions should be related.
This *initialization* happens once before the loop starts.
The loop is executed while this *condition* is true.
This *update* is executed after each iteration.
The variable i is defined only in this for loop.
This loop executes 6 times.

- Use a `for` loop when a variable runs from a starting value to an ending value with a constant increment or decrement

- for Loop Examples

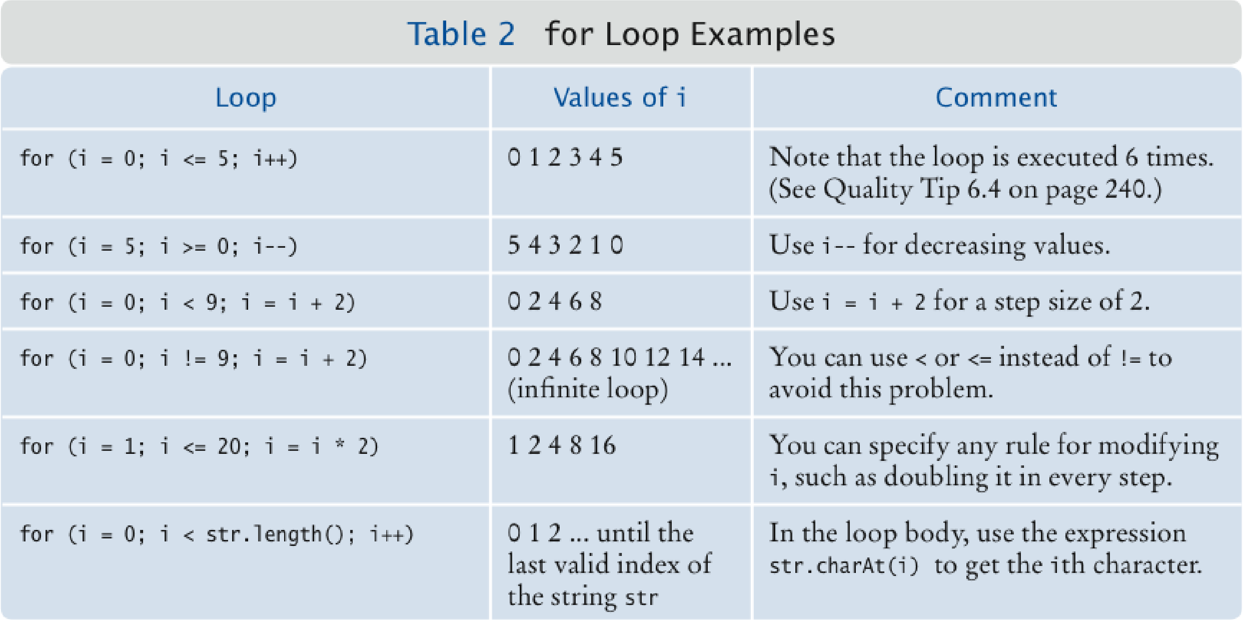


Table 2 for Loop Examples

Loop	Values of i	Comment
`for (i = 0; i <= 5; i++)`	0 1 2 3 4 5	Note that the loop is executed 6 times. (See Quality Tip 6.4 on page 240.)
`for (i = 5; i >= 0; i--)`	5 4 3 2 1 0	Use `i--` for decreasing values.
`for (i = 0; i < 9; i = i + 2)`	0 2 4 6 8	Use `i = i + 2` for a step size of 2.
`for (i = 0; i != 9; i = i + 2)`	0 2 4 6 8 10 12 14 … (infinite loop)	You can use < or <= instead of != to avoid this problem.
`for (i = 1; i <= 20; i = i * 2)`	1 2 4 8 16	You can specify any rule for modifying `i`, such as doubling it in every step.
`for (i = 0; i < str.length(); i++)`	0 1 2 … until the last valid index of the string `str`	In the loop body, use the expression `str.charAt(i)` to get the `i`th character.

Q. Rewrite the following for loop into a while loop and draw the flow chart.

```
int s = 0;
for (int i = 1; i <= 10; i++) s = s + i;
```

Solution:

```
int s = 0;
int i = 1;
while (i <= 10)
{
i++;
s = s + i;
}
```

Self Check 6.4

How many times does the following for loop execute?

```
for (i = 0; i <= 10; i++)
   System.out.println(i * i);
```

Answer: 11 times.

Nested Loops

Nested Loops: Put loops together (loop inside loop)

Nested Loops Example:

1. Write a java program to print the following .

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

```
public class NestedLoop {
        public static void main(String[] args)
        {
        for (int i=1;i<=4;i++)
        {
              for(int j=1; j<=i; j++)
              {
                     System.out.print("*");
            }
                     System.out.println();
              }}}
```

2. Write a java program to print the following .

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

```
public class NestedLoop {
        public static void main(String[] args)
        {
        for (int i=1;i<=3;i++)
        {
              for(int j=1; j<=4; j++)
              {
                     System.out.print("*");
            }
                     System.out.println();
              }
}}
```

- **Debugger:** a program to execute your program and analyze its run-time behavior
- **A debugger: lets you stop and restart your program.**
- **The larger your programs, the harder to debug them simply by inserting print commands**
- Three key concepts of debugger:
  - *Breakpoints*
  - *Single-stepping*
  - *Inspecting variables*
- **In Debugging: Execution is suspended whenever a breakpoint is reached**
- **In a debugger, a program runs at full speed until it reaches a breakpoint.**
- When program terminates, debugger stops as well
- Breakpoints stay active until you remove them
- Two variations of single-step command:
  - ***Step Over****: Skips method calls*
  - ***Step Into****: Steps inside method calls*
- **Self Check 6.13**

In the debugger, you are reaching a call to System.out.println. Should you step into the method or step over it?

Answer: You should step over it because you are not interested in debugging the internals of the println method.

Chapter 7 – Arrays and Array Lists

Array: Sequence of values of the same type .

- Construct array:

```
new double[10]
```

- Store in variable of type `double[]`:

```
double[] data = new double[10];
```

Declaring Arrays:

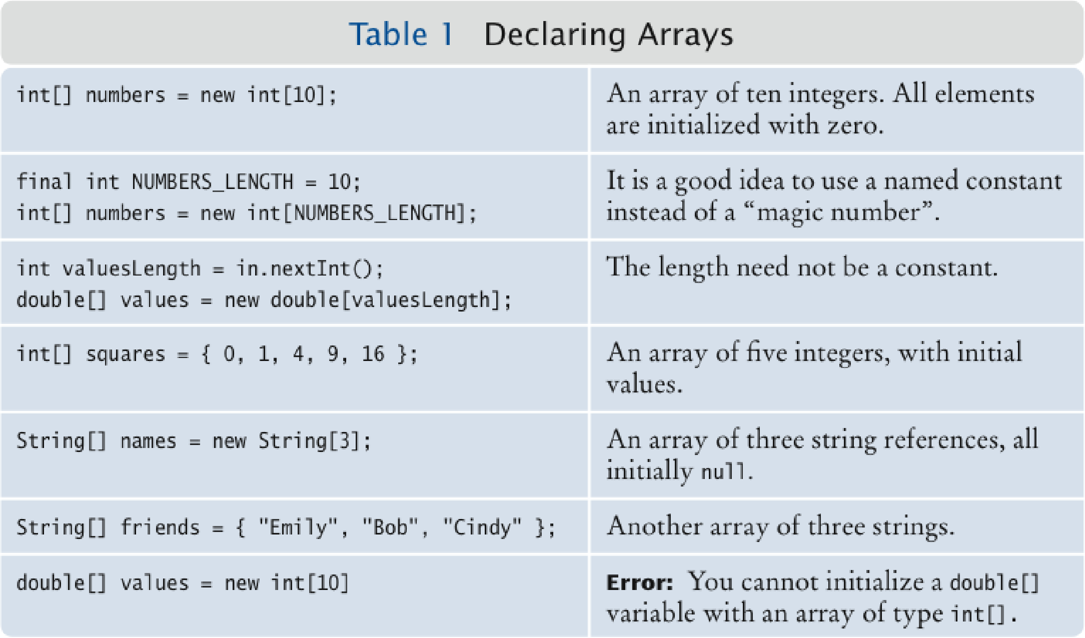


Table 1 Declaring Arrays

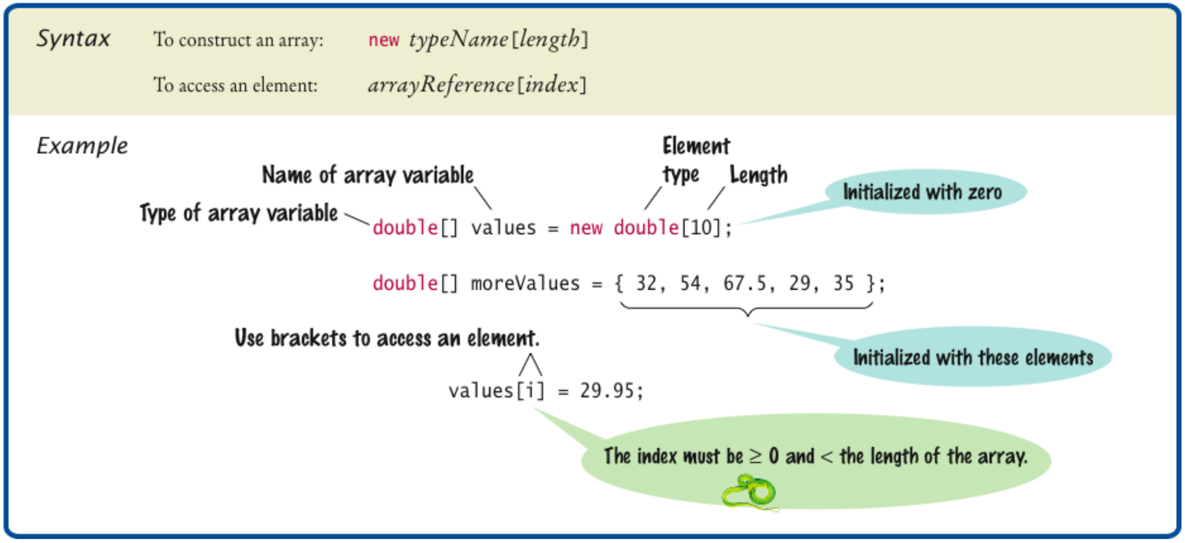
`int[] numbers = new int[10];`	An array of ten integers. All elements are initialized with zero.

`final int NUMBERS_LENGTH = 10;`
`int[] numbers = new int[NUMBERS_LENGTH];`
	It is a good idea to use a named constant instead of a “magic number”.

`int valuesLength = in.nextInt();`
`double[] values = new double[valuesLength];`
	The length need not be a constant.

`int[] squares = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`String[] names = new String[3];`	An array of three string references, all initially `null`.
`String[] friends = { "Emily", "Bob", "Cindy" };`	Another array of three strings.
`double[] values = new int[10]`	**Error:** You cannot initialize a `double[]` variable with an array of type `int[]`.

Syntax 7.1 Arrays



Syntax To construct an array: `new` *typeName*`[`*length*`]`

To access an element: *arrayReference*`[`*index*`]`

Example

```
double[] values = new double[10];

double[] moreValues = { 32, 54, 67.5, 29, 35 };

values[i] = 29.95;
```

Name of array variable
Element type
Length
Type of array variable
Initialized with zero
Initialized with these elements
Use brackets to access an element.
The index must be ≥ 0 and < the length of the array.

Self Check 7.1

What elements does the data array contain after the following statements?

```
double[] values = new double[10];
for (int i = 0; i < values.length; i++)
   values[i] = i * i;
```

Answer: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, but not 100

Get array length as `values.length`

Bounds error: Accessing a nonexistent element results.

Index values range: from `0` to `length - 1`

The first element index is; 0

The last element index is; array.length – 1

- Arrays have fixed length.

ArrayList

- ArrayList class: manages a sequence of objects.

- ArrayList class: Can grow and shrink as needed

- ArrayList class: supplies methods for many common tasks, such as inserting and removing elements

- ArrayList: is a **generic class**:

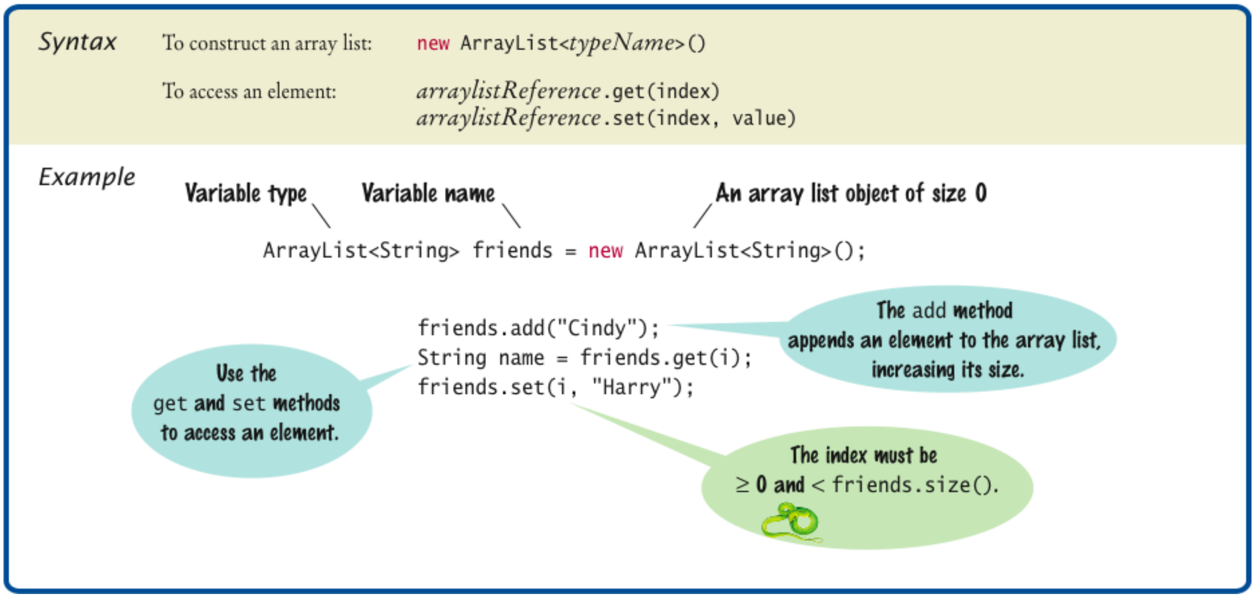
- **Size: number of elements in ArrayList**

- To obtain the value an element at an index, use the `get` method
- Index starts at 0
- `String name = names.get(2);`

```
// gets the third element of the array list
```

- Bounds error if index is out of range
- `add` method: add an object to the end of the array list.

- To **Replace** an element to a new value, use the **set** method.



Syntax To construct an array list: `new ArrayList<typeName>()`

To access an element:

```
arraylistReference.get(index)
arraylistReference.set(index, value)
```

Example

```
ArrayList<String> friends = new ArrayList<String>();

friends.add("Cindy");
String name = friends.get(i);
friends.set(i, "Harry");
```

Variable type
Variable name
An array list object of size 0
Use the get and set methods to access an element.
The add method appends an element to the array list, increasing its size.
The index must be ≥ 0 and < friends.size().

- ## Self Check 7.3

How do you construct an array of 10 strings? An array list of strings?

Answer:

```
new String[10];
new ArrayList<String>();
```

Self Check 7.4

What is the content of `names` after the following statements?

```
ArrayList<String> names = new ArrayList<String>();
names.add("A");
names.add(0, "B");
names.add("C");
```

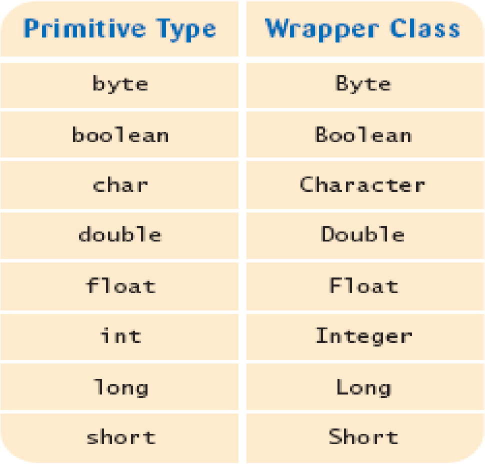
Answer: `names` contains the strings `"B"` and `"C"` at positions 0 and 1

Wrapper Classes:

- For each primitive type there is a **wrapper class** for storing values of that type:

```
Double d = new Double(29.95);
```

There are wrapper classes for all eight primitive types:



Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Q. What is the difference between the types double and Double?

Answer: double: is a primitive data type.

Double: is wrapper class that wraps the primitive data type double and makes it into an object.

- **Auto-boxing: Automatic conversion between primitive types and the corresponding wrapper classes.**
- **Auto-boxing even works inside arithmetic expressions**
- Storing wrapped numbers is quite inefficient
  - *Acceptable if you only collect a few numbers*
  - *Use arrays for long sequences of numbers or characters*

Self Check 7.5

What is the difference between the types `double` and `Double`?

Answer: `double` is one of the eight primitive types. `Double` is a class type.

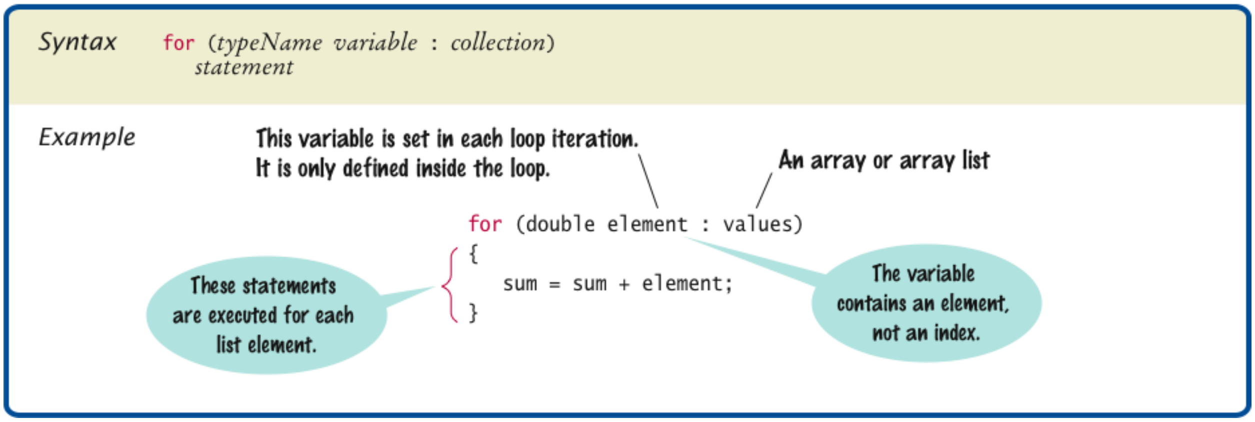
Self Check 7.6

Suppose `values` is an `ArrayList<Double>` of size > 0. How do you increment the element with index 0?

Answer:

```
values.set(0, values.get(0) + 1);
```

The “for each” Loop



Syntax

```
for (typeName variable : collection)
   statement
```

Example

```
for (double element : values)
{
   sum = sum + element;
}
```

This variable is set in each loop iteration. It is only defined inside the loop.
An array or array list
These statements are executed for each list element.
The variable contains an element, not an index.

Q. Rewrite the following loops without using the “for each” construct.

```
double[] values = ...;
  double sum = 0;
  for (double element : values)
  {
     sum = sum + element;
  }
```

Solution: Using Traditional `for` Loop

```
double[] values = ...;
  double sum = 0;
  for (int i = 0; i < values.length; i++)
  {
     double element = values[i];
     sum = sum + element;
  }
```

- The “for each loop” does not allow you to modify the contents of an array:

Self Check 7.7

Write a “for each” loop that prints all elements in the array `values`.

Answer:

```
for (double element : values)
    System.out.println(element);
```

Self Check 7.8

What does this “for each” loop do?

```
int counter = 0; for (BankAccount a : accounts)
{
   if (a.getBalance() == 0) { counter++; }
}
```

Answer: It counts how many accounts have a zero balance.

- Usually, array is partially filled

Self Check 7.9

Write a loop to print the elements of the partially filled array `values` in reverse order, starting with the last element.

Answer:

```
for (int i = valuesSize - 1; i >= 0; i--)
   System.out.println(values[i]);
```

Self Check 7.10

How do you remove the last element of the partially filled array `values`?

Answer:

```
valuesSize--;
```

Self Check 7.11

Why would a programmer use a partially filled array of numbers instead of an array list?

Answer: You need to use wrapper objects in an `ArrayList<Double>`, which is less efficient.

- Fill an array with zeroes:

```
for (int i = 0; i < values.length; i++)
{
   values[i] = 0;
}
```

- Fill an array list with squares (0, 1, 4, 9, 16, ...):

```
for (int i = 0; i < values.size(); i++)
{
   values.set(i, i * i;
}
```

- To compute the sum of all elements, keep a running total:

```
double total = 0;
for (double element : values)
{
   total = total + element;
}
```

- To obtain the average, divide by the number of elements:

```
double average = total /values.size();
// for an array list
```

linear search: The process of checking all elements until you have found a match

Removing an Element from array list

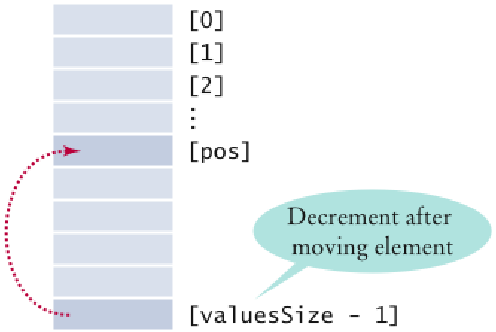


Figure 9
Removing an Element in an Unordered Array

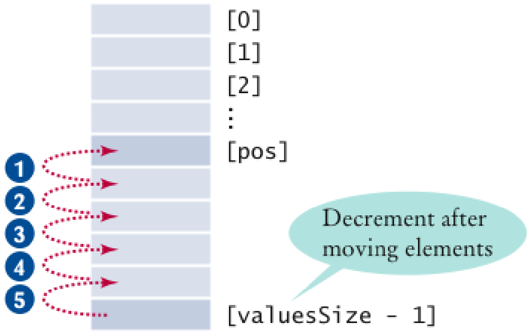


Figure 10
Removing an Element in an Ordered Array

- Array list ⇒ use method `remove`
- Unordered array ⇒
    1. *Overwrite the element to be removed with the last element of the array*
    2. *Decrement the variable tracking the size of the array*

```
values[pos] = values[valuesSize - 1];
valuesSize--;
```

- Ordered array ⇒
    1. *Move all elements following the element to be removed to a lower index*
    2. *Decrement the variable tracking the size of the array*

```
for (int i = pos; i < valuesSize - 1; i++)
{
   values[i] = values[i + 1];
}
valuesSize--;
```

Inserting an Element from array list

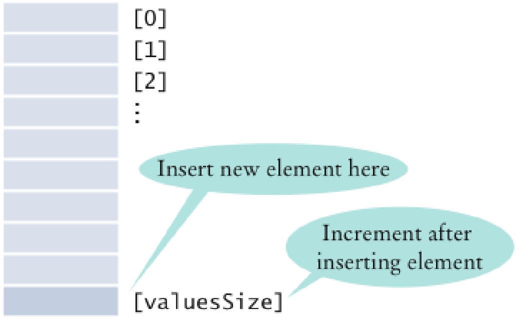


Figure 11
Inserting an Element in an Unordered Array

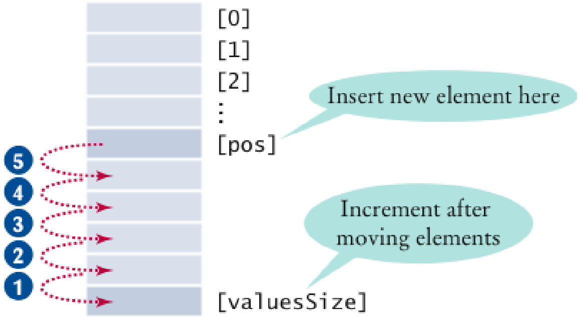


Figure 12
Inserting an Element in an Ordered Array

- Array list ⇒ use method `add`
- Unordered array ⇒
    1. *Insert the element as the last element of the array*
    2. *Increment the variable tracking the size of the array*

```
if (valuesSize < values.length)
{
   values[valuesSize] = newElement;
   valuesSize++;
}
```

- Ordered array ⇒
    1. *Start at the end of the array, move that element to a higher index, then move the one before that, and so on until you finally get to the insertion location*
    2. *Insert the element*
    3. *Increment the variable tracking the size of the array*

```
if (valuesSize < values.length)
{
   for (int i = valuesSize; i > pos; i--)
   {
      values[i] = values[i - 1];
   }
   values[pos] = newElement;
   valuesSize++;
}
```

- To make a true copy of an array, call the `Arrays.copyOf` method:

  ```
  double[] prices = Arrays.copyOf(values, values.length);
  ```

- To grow an array that has run out of space, use the `Arrays.copyOf` method:

  ```
      values = Arrays.copyOf(values, 2 * values.length);
  ```

Self Check 7.12

What does the `find` method do if there are two bank accounts with a matching account number?

Answer: It returns the first match that it finds.

Self Check 7.13

Would it be possible to use a “for each” loop in the `getMaximum` method?

Answer: Yes, but the first comparison would always fail.

Regression Testing

- **Test suite:** a set of tests for repeated testing
- **Cycling:** bug that is fixed but reappears in later versions
- **Regression testing:** repeating previous tests to ensure that known failures of prior versions do not appear in new versions

Self Check 7.16

Suppose you modified the code for a method. Why do you want to repeat tests that already passed with the previous version of the code?

Answer: It is possible to introduce errors when modifying code.

Self Check 7.17

Suppose a customer of your program finds an error. What action should you take beyond fixing the error?

Answer: Add a test case to the test suite that verifies that the error is fixed.

Self Check 7.19

How do you declare and initialize a 4-by-4 array of integers?

Answer:

```
int[][] array = new int[4][4];
```

Chapter 8 – Designing Classes

Discovering Classes

- A class represents a single concept from the problem domain
- Name for a class should be a noun that describes concept
- **Actors:** (end in -er, -or) – objects do some kinds of work for you: Scanner
- **Utility classes** – no objects, only static methods and constants: Math
- **Program starters**: only have a main method

Self Check 8.1

What is the rule of thumb for finding classes?

Answer: Look for nouns in the problem description.

Coupling and Cohesion

What is the differace between coupling and cohesion?

- **Cohesion:** A class should represent a single concept.
- The public interface of a class is *cohesive* if all of its features are related to the concept that the class represents

- **Coupling:** A class depends on another if it uses objects of that class.

Example for cupling:

- CashRegister depends on Coin to determine the value of the payment

- Coin does not depend on CashRegister

- **UML: Unified Modeling Language**
- **High coupling = Many class dependencies**

Self Check 8.4

Why does the `Coin` class not depend on the `CashRegister` class?

Answer: None of the `Coin` operations require the `CashRegister` class.

Self Check 8.5

Why should coupling be minimized between classes?

Answer: If a class doesn't depend on another, it is not affected by interface changes in the other class.

- **Accessor:** Does not change the state of the implicit parameter:
- **Mutator:** Modifies the object on which it is invoked:
- **Immutable class:** Has no mutator methods (e.g., String):

Self Check 8.6

Is the `substring` method of the `String` class an accessor or a mutator?

Answer: It is an accessor — calling `substring` doesn't modify the string on which the method is invoked. In fact, all methods of the `String` class are accessors.

Self Check 8.7

Is the `Rectangle` class immutable?

Answer: No — `translate` is a mutator.

- **Side effect of a method:** Any externally observable data modification.
- Modifying explicit parameter can be surprising to programmers

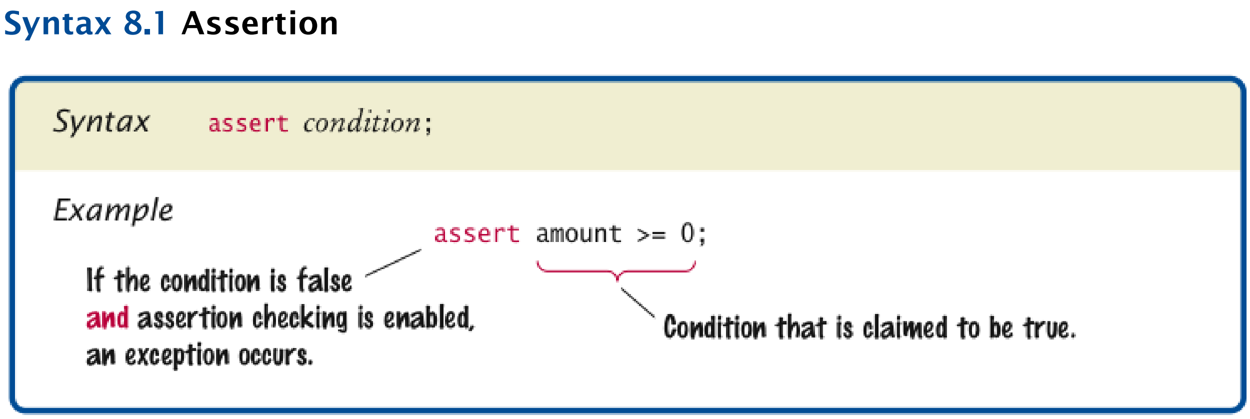
Call by Value and Call by Reference

- **Call by value:** Method parameters are copied into the parameter variables when a method starts
- **Call by reference:** Methods can modify parameters
- Java has call by value
- A method can change state of object reference parameters, but cannot replace an object reference with another

Preconditions

- **Precondition: Requirement that the caller of a method must meet.**

- **If precondition is violated, method is not responsible for computing the correct result. It is free to do *anything***



Syntax 8.1 Assertion

Syntax `assert condition;`

Example

```
assert amount >= 0;
```

If the condition is false and assertion checking is enabled, an exception occurs.

Condition that is claimed to be true.

- **Postcondition:** requirement that is true after a method has completed .
- If method call is in accordance with preconditions, it must ensure that postconditions are valid
- There are two kinds of postconditions:
  - *The return value is computed correctly*
  - *The object is in a certain state after the method call is completed*

- Contract: If caller fulfills preconditions, method must fulfill postconditions .

Self Check 8.10

Why might you want to add a precondition to a method that you provide for other programmers?

Answer: Then you don’t have to worry about checking for invalid values — it becomes the caller’s responsibility.

Static Methods

Static Methods : Every method must be in a class
Static Methods : is not invoked on an object .

- *Numbers aren't objects, you can't invoke methods on them. E.g.* `x.sqrt()` *can never be legal in Java*

- `main` is static — there aren't any objects yet

Self Check 8.12

Suppose Java had no static methods. How would you use the `Math.sqrt` method for computing the square root of a number *x*?

Answer:

```
Math m = new Math();
y = m.sqrt(x);
```

Static variable: belongs to the class, not to any object of the class.

Static variables: should always be declared as private.

- Minimize the use of static variables.

Self Check 8.14

Name two static variables of the `System` class.

Answer: `System.in` and `System.out`.

Scope of Local Variables

- **Scope of variable:** Region of program in which the variable can be accessed
- Scope of a local variable extends from its declaration to end of the block that encloses it
- Scope of a local variable cannot contain the definition of another variable with the same name:

Overlapping Scope

- A local variable can shadow a variable with the same name
- Local scope wins over class scope.
- Access shadowed variables by qualifying them with the `this` reference:

```
value = this.value * exchangeRate;
```

- Generally, shadowing an instance variable is poor code — error-prone, hard to read

Packages

- **Package:** **Set of related classes.**
- **Important packages in the Java library:**

Package	Purpose	Sample Class
`java.lang`	Language support	`Math`
`java.util`	Utilities	`Random`
`java.io`	Input and output	`PrintStream`
`java.awt`	Abstract Windowing Toolkit	`Color`
`java.applet`	Applets	`Applet`
`java.net`	Networking	`Socket`
`java.sql`	Database Access	`ResultSet`
`javax.swing`	Swing user interface	`JButton`
`omg.w3c.dom`	Document Object Model for XML documents	`Document`

- To put classes in a package, you must place a line

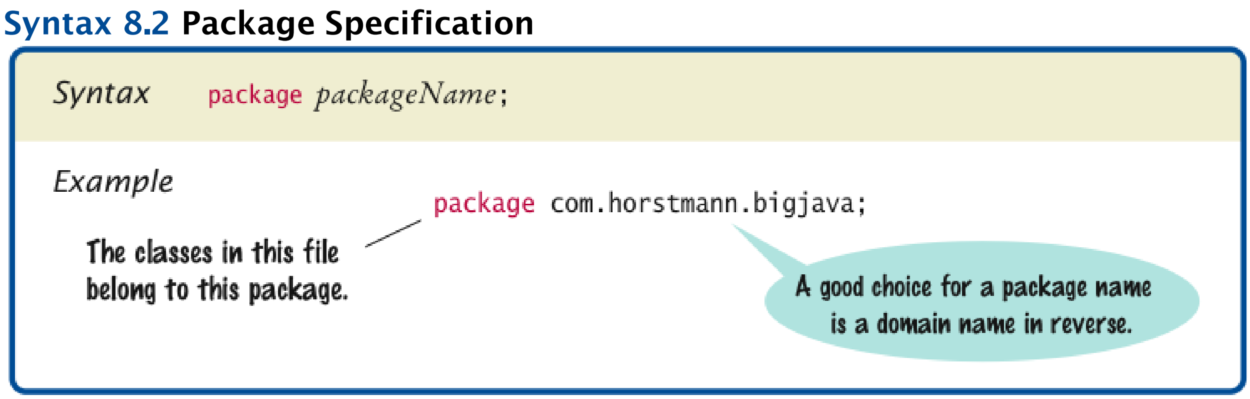
```
package packageName;
```

- Package name consists of one or more identifiers separated by periods
- For example, to put the `Financial` class introduced into a package named `com.horstmann.bigjava`, the `Financial.java` file must start as follows:

```
package com.horstmann.bigjava;

public class Financial
{
   ...
}
```

- Default package has no name, no `package` statement



Syntax 8.2 Package Specification

Syntax `package packageName;`

Example

```
package com.horstmann.bigjava;
```

The classes in this file belong to this package.

A good choice for a package name is a domain name in reverse.

Importing Packages

- Can always use class without importing:

  ```
  java.util.Scanner in = new java.util.Scanner(System.in);
  ```
- Tedious to use fully qualified name
- Import lets you use shorter class name:

  ```
  import java.util.Scanner;
  ...
  Scanner in = new Scanner(System.in)
  ```
- Can import all classes in a package:

  ```
  import java.util.*;
  ```
- Never need to import `java.lang`
- You don’t need to import other classes in the same package
- Use packages to avoid name clashes
- Package names should be unambiguous
- Recommendation: start with reversed domain name:

  ```
  com.horstmann.bigjava
  ```
- `edu.sjsu.cs.walters`: for Britney Walters’ classes (`walters@cs.sjsu.edu`)
- Path name should match package name:

  ```
  com/horstmann/bigjava/Financial.java
  ```

- **Base directory:** holds your program's Files.
- Path name, relative to base directory, must match package name.

Self Check 8.18

Which of the following are packages?

a. `java`
b. `java.lang`
c. `java.util`
d. `java.lang.Math`

Answer:

a. *No*
b. *Yes*
c. *Yes*
d. *No*

Self Check 8.19

Is a Java program without `import` statements limited to using the default and `java.lang` packages?

Answer: No — you simply use fully qualified names for all other classes, such as `java.util.Random` and `java.awt.Rectangle`

- Unit test frameworks: simplify the task of writing classes that contain many test cases.
- whenever you implement a class, also make a companion test class. Run all tests whenever you change your code