Chapter 3
Writing Simple Classes
Chapter 3
Writing Simple Classes
Using BlueJ
BlueJ IDE

- IDE (integrated development environment)
- Learning tool for understanding the three fundamental concepts of object-oriented programming (OOP)

- classes
- objects
- methods
What is a Java program?

- A Java program consists of one or more classes that are used to create objects.
- Each class defines a different kind of object.
- Each class defines the functionality or the behaviour of its objects by defining a number of methods (functions) that an object can execute.
BlueJ Project Window

Each yellow box represents a class.

Each class has a name.

To edit a class double click on its yellow box. This will bring up an editor containing the Java source code for the class. If the yellow box is shaded then the class needs to be compiled.
Constructing an object (1)

Right click on box and choose a constructor

The object workbench
Constructing an object (2)

Choose a name for the object

Enter data required to construct the object. Here it is just the radius.
Constructing an object (3)

The object workbench

The object is shown in red on the object workbench
Invoking methods on an object

Right click on the object to get the method menu

remove the object

show the data fields
Method result

Choose `getArea` method

Area of circle of radius 2
Multiple objects

Three CircleCalculator objects for radii 2, 3, and 4
Each object has its own set of instance data fields.

Instance data fields:

Each object has its own set of instance data fields.
The anatomy of a class

public class CircleCalculator
{

(1) Instance data fields
(2) Constructor declarations
(3) Method declarations

}
private double radius;
private double area;
private double circumference;

Instance data fields are variables associated with an object

Each CircleCalculator object has its own set of instance data fields
public CircleCalculator(double r) {
    radius = r;
    area = Math.PI * radius * radius;
    circumference = 2.0 * Math.PI * radius;
}

This constructor initializes the three instance data fields using the radius supplied as an actual argument.
CircleCalculator methods (1)

```java
public double getArea()
{
    return area;
}
```

- **Method declaration**: The first line of the method declaration is called the method prototype.
- **Method name**: `getArea`
- **Return type**: `double`

This method returns the value of the instance data field for the area of the circle as the value of the method.
public double getRadius() {
    return radius;
}

public double getCircumference() {
    return circumference;
}
Triangle calculator problem

- **Problem:** Given the length of two sides of a triangle and the contained angle in degrees, compute the third side length, the other two angles, and the perimeter of the triangle.

- Step 1: we need triangle math formulas
- Step 2: write the class
- Step 3: test the class
Triangle formulas

\[ c = \sqrt{a^2 + b^2 - 2ab \cos \gamma} \]
\[ \alpha = \cos^{-1} \left( \frac{b^2 + c^2 - a^2}{2bc} \right) \]
\[ \beta = \cos^{-1} \left( \frac{c^2 + a^2 - b^2}{2ca} \right) \]

perimeter = \(a + b + c\)

\[ s = \text{perimeter} / 2 \]

area = \(\sqrt{s(s-a)(s-b)(s-c)}\)
Creating an object

Right clicking the TriangleCalculator box gives this dialog for creating an object.

The TriangleCalculator constructor has three arguments for the two sides and the contained angle.
TriangleCalculator methods

```
Inherited from Object

double checkAngleSum()
double getA()
double getAlpha()
double getArea()
double getB()
double getBeta()
double getC()
double getGamma()
double getPerimeter()

Inspect
Remove
```

Instance data fields (next slide)
Instance data fields (inspect)

Object of class TriangleCalculator (triangle)  

Instance fields
private double a = 1.0
private double b = 1.0
private double c = 1.414213562373095
private double alpha = 45.00000000000001
private double beta = 45.00000000000001
private double gamma = 90.0
private double perimeter = 3.414213562373095
private double area = 0.5
public class TriangleCalculator
{
    private double a, b, c;
    private double alpha;
    private double beta;
    private double gamma;
    private double double perimeter, area;
public TriangleCalculator( double sideA, double sideB, double g) {
    double s;
    a = sideA;
    b = sideB;
    c = Math.sqrt(a*a + b*b - 2*a*b*Math.cos(Math.toRadians(g)));
}
alpha = Math.acos( (b*b + c*c - a*a) / (2*b*c) );
alpha = Math.toDegrees(alpha);
beta = Math.acos( (c*c + a*a - b*b) / (2*c*a) );
beta = Math.toDegrees(beta);
gamma = g;
TriangleCalculator (4)

```java
    perimeter = a + b + c;
    s = perimeter / 2;
    area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
}
```

End of constructor
TriangleCalculator (5)

```java
public class TriangleCalculator {
    public double getA() {
        return a;
    }

    public double checkAngleSum() {
        return alpha + beta + gamma;
    }
}
```

End of class declaration

for testing

8 get methods
Roots of quadratic equation

\[ ax^2 + bx + c = 0 \]

\[
r_1 = \frac{1}{2a} \left( -b - \sqrt{b^2 - 4ac} \right) \]

\[
r_2 = \frac{1}{2a} \left( -b + \sqrt{b^2 - 4ac} \right) \]

**PROBLEM**
Given the coefficients a, b, and c of a quadratic equation find the real roots, if any.
Creating an object

Right clicking the QuadraticRootFinder box gives this dialog for creating an object.

The QuadraticRootFinder constructor has three arguments for the coefficients of the quadratic equation.
QuadraticRootFinder methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inherited from Object</td>
<td></td>
</tr>
<tr>
<td>double getA()</td>
<td></td>
</tr>
<tr>
<td>double getB()</td>
<td></td>
</tr>
<tr>
<td>double getC()</td>
<td></td>
</tr>
<tr>
<td>doublegetRoot1()</td>
<td></td>
</tr>
<tr>
<td>doublegetRoot2()</td>
<td></td>
</tr>
<tr>
<td>void setA(value)</td>
<td></td>
</tr>
<tr>
<td>void setB(value)</td>
<td></td>
</tr>
<tr>
<td>void setC(value)</td>
<td></td>
</tr>
</tbody>
</table>

*Inspect*

*Remove*

![BlueJ: Method Result](image1)

**double result = 0.5**

Close

![BlueJ: Method Result](image2)

**double result = 1.5**

Close
Changing a coefficient

```
// Change the value of the constant coefficient.
// @param value the new value for the constant coefficient.
void setC(double value)

rootFinder.setC(1)
```

```
Object of class QuadraticRootFinder (rootFinder)  Show static fields

Instance fields
private double a = 1.0
private double b = -2.0
private double c = 1.0
private double root1 = 1.0
private double root2 = 1.0

Close
```
public class QuadraticRootFinder {
    private double a, b, c;
    private double root1, root2;

    instance data fields
public QuadraticRootFinder(
    double a, double b, double c)
{
    this.a = a;
    this.b = b;
    this.c = c;
    doCalculations();
}

"this" distinguishes the private data fields of this object from a local variable of the same name.

A local method call expression (calling a method in the same class)
private void doCalculations() {
    double d = Math.sqrt(b*b - 4*a*c);
    root1 = (-b - d) / (2.0 * a);
    root2 = (-b + d) / (2.0 * a);
}

method does not return a value

Not available outside class

We will call this method four times

local variable

Instance data fields. must not be declared here again
public double getRoot1()
{
    return root1;
}

public double getRoot2()
{
    return root2;
}
public double getA()
{
    return a;
}

public double getB()
{
    return b;
}

public double getC()
{
    return c;
}
public void setA(double value)
{
    a = value;
    doCalculations();
}

public void setB(double value)
{
    b = value;
    doCalculations();
}

public void setC(double value)
{
    c = value;
    doCalculations();
}

Set methods are used to change the value of an instance data field after the object has been constructed.
private double root1;
private double root2;
...
public void doCalculations()
{
    ...
    double root1 = ...  
    double root2 = ...  
}

These are different variables than the instance data fields.

The local variables here are said to shadow the instance data fields and this is almost NEVER what you want.

these variables will NEVER be assigned values so they will always have their default value of 0
Chapter 3
Writing Simple Classes

Using BeanShell with Objects
OOP Syntax using BeanShell

- In BlueJ the basic OOP operations are done interactively using the mouse and dialog boxes to
  - construct objects
  - Choose a method to invoke on an object (sending a message to an object)
  - See the results of invoking a method
- We can illustrate the actual Java syntax for these operations using BeanShell.
Syntax for creating objects

CircleCalculator circle1 =
    new CircleCalculator(2.0);

TriangleCalculator triangle =
    new TriangleCalculator(1,1,90);

QuadraticRootFinder rootFinder =
    new QuadraticRootFinder(1,-2,0.75);

General syntax is

ClassName objectName =
    new ClassName(actualArgumentList);
Method call expressions

circle1.getArea()
triangle.getC()
triangle.setC(1.0)
rootFinder.getRoot1()

General syntax is

objectName.methodName(actualArgumentList);
Assigning results to variables

double result = circle1.getArea();

double c = triangle.getC();

triangle.setC(1.0);

double root1 = rootFinder.getRoot1();

triangle.setC(1.0) doesn't return a result. It just changes the value of the coefficient c in the quadratic equation.
Displaying results

- BlueJ
  - Method result box
  - Use inspect menu item from object menu

- BeanShell
  - use print
  - use show for automatic display

- Java
  - System.out.print, System.out.println
  - GUI methods using TextField, TextArea
```java
addClassPath("c:/book-projects/chapter3");
CircleCalculator circle1 =
    new CircleCalculator(2.0);
double area1 = circle1.getArea();
print(area);
12.566370614359172
double circum1 = circle1.getCircumference();
print(circum1);
12.566370614359172
```
addClassPath("c:/book-projects/chapter3");
CircleCalculator circle2 =
    new CircleCalculator(3.0);
CircleCalculator circle3 =
    new CircleCalculator(4.0);
double area2 = circle2.getArea();
double area3 = circle3.getArea();
double averageArea =
    (area1 + area2 + area3)/3;
print(averageArea);
30.368728984701335
addClassPath("c:/book-projects/chapter3");

TriangleCalculator triangle =
    new TriangleCalculator(1,1,90);

double c = triangle.getC();
print(c);
1.414213562373095
double angleSum = triangle.checkAngleSum();
print(angleSum);
180.0;
addClassPath("c:/book-projects/chapter3");
QuadraticRootFinder rootFinder =
    new QuadraticRootFinder(1,-2,0.75);
double r1 = rootFinder.getRoot1();
double r2 = rootFinder.getRoot2();
print(r1);
0.5
print(r2);
1.5
rootFinder.setC(1);
r1 = rootFinder.getRoot1();
r2 = rootFinder.getRoot2();
print(r1);
1.0
print(r2);
1.0
Ordinary Comments

- Single line comments

```java
private double gamma; // angle opposite side c
```

- Multi-line comments

```java
/* This private method is used in the
 * constructor and the three set methods
 * in order to update the roots in case
 * a coefficient is changed
 */
private void doCalculations()
...
/** Constructor for an object with specified radius.
 * @param r the radius of the circle
 */

public CircleCalculator(double r) {
    ...
}

Special Javadoc tag

They are just ordinary comments that begin with /**

Javadoc comments
Javadoc rules

- Use Javadoc block comment immediately before every class declaration to give a description of the class.
- Use a Javadoc comment immediately before each public constructor and method declaration.
- Use the Javadoc tags such as `@author`, `@version`, `@param`, `@return`
Javadoc tags

- **@author** *text*
  - *text* should specify the author of the class

- **@version** *text*
  - *text* should specify the version number

- **@param** *name* *text*
  - *name* is the name of a formal argument
  - *text* is a description of the argument

- **@return** *text*
  - description of return value of a method
QuadraticRootFinder example (1)

```java
/**
 * An object of this class can calculate ...
 * @author Reginald Hill
 * @version 1.1
 */

public class QuadraticRootFinder {
    // Instance data fields

    private double a, b, c;
    private double root1, root2;
```
/**
 * Construct a quadratic equation root finder given the coefficients.
 * @param a 1st coeff in \( ax^2 + bx + c \)
 * @param b 2nd coeff in \( ax^2 + bx + c \)
 * @param c 3rd coeff in \( ax^2 + bx + c \)
 */

public QuadraticRootFinder(double a,
               double b, double c)
{
    ...
}
This is an ordinary comment since the method is private

/**
 * This private method is used in the constructor and the three set methods in order to update the roots in case a coefficient is changed.
 */

private void doCalculations()
{
    ... 
}

/**
 * Return the first root.
 * @return the first real root or NaN if there are none.
 */
public double getRoot1()
{
    ...
}
/**
 * Change value of coefficient of \(x^2\).
 * @param value the new value for the
 * coefficient of \(x^2\)
 */

public void setA(double value)
{
    ... 
}
Other javadoc examples

- See the following classes from Chapter3

- CircleCalculator.java

- TriangleCalculator.java
Java style guide

- We have a style guide for this course.
- You can find it at

  http://www.cs.laurentian.ca/badams/c1046/style.html

- Assignments, beginning with the second assignment will be marked for style.
Viewing docs for a class (1)

In the editor window toggle between implementation (Source code) and Interface (javadoc)
Class interface

Constructor Summary

```
CircleCalculator(double r)
Constructor for an object with specified radius.
```

Method Summary

```
double getArea()
Return the area of the circle.

double getCircumference()
Return the circumference of the circle.

double getRadius()
Return the radius of the circle.
```
Viewing docs for a project (1)

Combined docs for all classes in the project
Viewing docs for a project (2)

combined docs for all three classes in the project
Syntax errors

- Forgetting a semi-colon
- Using undeclared variable
- Multiply declared variables
- Misspelling constructor name
- Forgetting to use new
Forgetting a semi-colon

```java
public class CircleCalculator {
    private double radius;
    private double area;
    private double circumference;
}
```

Syntax error: There is a semicolon missing at the end of a line.
Undeclared variable

```
// double s;

a = sideA;
b = sideB;
c = Math.sqrt(s*a + b*b - 2*a*b*Math.cos(Math.toRadians(c)));

// Angle opposite side a, contained by sides b and c

alpha = Math.acos((b*b + c*c - a*a) / (2*b*c));
alpha = Math.toDegrees(alpha);

// Angle opposite side b, contained by sides c and a

beta = Math.acos((c*c + a*a - b*b) / (2*a*c));
beta = Math.toDegrees(beta);

gamma = c;

// Calculate perimeter and use Heron's formula for
// the area in terms of the side lengths

double perimeter = a + b + c;
s = perimeter / 2;

area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
```

Message:
```
cannot resolve symbol
symbol : variable s
location: class TriangleCalculator

s = perimeter / 2;
^

You are using a symbol here (a name for a variable, a method, or a class) that has not been declared in any visible scope. Check the spelling of that name - did you mistype it? Or did you forget to declare it? Or maybe you did declare it, but it is not visible from here.
```
Multiply declared variable

```java
public class TriangleCalculator {
    public static void main(String[] args) {
        double s;
        a = sideA;
        b = sideB;
        c = Math.sqrt(a*a + b*b - 2*a*b*Math.cos(Math.toRadians(g)));
        // Angle opposite side a, contained by sides b and c
        alpha = Math.acos((b*b + c*c - a*a) / (2*b*c));
        alpha = Math.toDegrees(alpha);
        // Angle opposite side b, contained by sides c and a
        beta = Math.acos((c*c + a*a - b*b) / (2*c*a));
        beta = Math.toDegrees(beta);
        gamma = g;
        // Calculate perimeter and use Heron's formula for
        // the area in terms of the side lengths
        double perimeter = a + b + c;
        double s = perimeter / 2;
        area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
    }
}
```

Message:

There is already a variable (or maybe a parameter) in this method that has the same name. Use a different name for this one. (Or maybe you meant to use the same variable here? Then remove the type name here so that it does not look like a new declaration.)
Misspelled constructor name

```java
public class CircleCalculator {
    // Instance data fields defining a CircleCalculator object
    private double radius;
    private double area;
    private double circumference;

    /**
     * Constructor for an object with specified radius.
     * @param r the radius of the circle
     */
    public circleCalculator(double r) {
        radius = r;
        area = Math.PI * radius * radius;
        circumference = 2.0 * Math.PI * radius;
    }
}
```

A method declaration must have a declared return type. For example, if your method returns a String, write:

```java
public String myMethod();
```

If you do not want to return a value from this method, use the special word "void" to indicate that there is no return type. For example:

```java
public void myMethod();
```
BeanShell error

```java
addClassPath("c:/book-projects/chapter3");
CircleCalculator circle = CircleCalculator(2.0);
// Error: Typed variable declaration :
// Command not found: CircleCalculator
// <at unknown location>
```

Here we forgot to use `new` to construct an object so BeanShell is assuming that `CircleCalculator(2.0)` is a method call expression and there is no method with the name `CircleCalculator`
Logical errors

Using an incorrect formula
Redefining an instance variable
Using return type on a constructor
Invoking a method on non-existent object
What are logical errors

- Syntax errors occur at compile-time
- Logical errors occur at run-time
- Logical errors are usually harder to find than syntax errors
- Logical errors give a run-time error message
- Fixing logical errors is called debugging
Using an incorrect formula

- This is a very common error which can be uncovered only by testing your program to see if it produces correct results in the test cases.

- Example: using a + sign instead of - sign
Redeclaring an instance variable

- See slide 38 on shadowing.
- Example: In CircleCalculator replace

```java
radius = r;
area = Math.PI * radius * radius;
circumference = 2.0 * Math.PI * radius;
```

by

```java
double radius = r;
double area = Math.PI * radius * radius;
double circumference = 2.0 * Math.PI * radius;
```
Using return type on constructor

- Replace the CircleCalculator constructor by

```java
public void CircleCalculator(double r) {
    ...
}
```

- In BlueJ try right click on the yellow class triangle and you will not find the constructor entry new CircleCalculator(r). Instead you will find new CircleCalculator() and you will not be asked to enter a radius
**Explanation**

- public void CircleCalculator(double r) no longer declares a constructor.
- It is a strange method with the same name as the constructor and a void return type (not illegal)
- Therefore you have declared no constructors in the class and the compiler has provided a default one with no arguments which appears on the menu (see Chapter 4).
Non-existent objects (BeanShell)

```java
addclassPath("c:/book-projects/chapter3");
CircleCalculator circle;
double area = circle.getArea();
```

Here we have declared `circle` as a variable of type `CircleCalculator` but we have not constructed an object to assign to it. Then `circle.getArea()` is a method invocation on a non-existent object called a null object and you will get an error message called a "null pointer exception". The error can be fixed by using a statement such as

```java
circle = new CircleCalculator(2.0);
```
Summary of Terminology (1)

- **Syntax or grammar**
  - rules defining the legal statements in a language

- **Simple identifier**
  - radius, doCalculations, circle1, triangle

- **Type**
  - **primitive type:** (int, float, double)
  - **object type:** (CircleCalculator)
Summary of Terminology (2)

- **Class:**
  - A definition of a set of objects of a specific type and their behaviour.
  - *CircleCalculator, TriangleCalculator*

- **Class:**
  - A home for some function not associated with any objects
  - *Math*
Summary of Terminology (3)

- **Object:**
  - An entity that has identity (a name), state, and behaviour.

- **Instance:**
  - another name for an object

- **Examples:**
  - CircleCalculator object called circle1
  - TriangleCalculator object called triangle
Summary of Terminology (4)

**Method:**
- A function or operation defined in a class that can be invoked on an object of the class. Such methods are often called *instance methods*

**Examples**
- `getArea`
- `setA`
- `checkAngleSum`
- `doCalculations`

We will learn about static methods later
Summary of Terminology (5)

- **Constructor**:
  - A special kind of method defined in a class that is used to create an object (instance of the class) having specified properties.
  - A constructor must have same name as its class and has no return type (this distinguishes it from a method which always has a return type (e.g. `double` or `void`).

- **Example**: CircleCalculator
Summary of Terminology (6)

- **Variable declaration:**
  - A statement having one of the forms
  - `accessModifier typeName identifier;`
  - `accessModifier typeName identifier = expression;`

- **Examples:**
  - `private double radius;`
  - `double radius;`
  - `double radius = 2.0;`
**Summary of Terminology (7)**

- **Object example:**
  - `CircleCalculator circle1 = new CircleCalculator(2.0);`

- **Multiple declarations:**
  - The definition can be extended to include multiple declarations of the same type in a single statement:
    - `int n = 123, remainder, hundreds, tens, units;`
    - `double area, circumference;`
Summary of Terminology (8)

- **Arithmetic expression:**
  - an expression involving variables and operators that evaluates to a numeric value.

- **Examples:**
  - radius;
  - 2.0 * Math.PI * radius;
  - circle1.getArea() + circle2.getArea();
Summary of Terminology (9)

- **Assignment statement:**
  - A statement of form
  - `identifier = expression;

- **Examples:**
  - `radius = 2.0;`
  - `circle1 = new CircleCalculator(2.0);`
  - `area = Math.PI * radius * radius;`
Summary of Terminology (10)

- Class declaration (not the most general)

```java
public class ClassName {
    Data field declarations
    Constructor declarations
    Method declarations
}
```
Class declaration examples

```java
public class CircleCalculator {
    ...
}

public class QuadraticRootFinder {
    ...
}
```
Constructor declaration (not most general)

```java
modifiers ClassName ( formalArgumentList )
{
  Local declarations and other statements
}
```
Constructor declaration examples

```java
public CircleCalculator(double r)
{
    ...
}
```

```java
public QuadraticRootFinder(
    double a, double b, double c)
{
    ...
}
```

Summary of Terminology (12)

- Method declaration (not most general)

```java
modifiers returnType ClassName ( formalArgumentList )
{
    Local declarations and other statements
}
```
Method declaration examples

```java
public void setC(double value)
{
    ...
}

private void doCalculations()
{
    ...
}

public double getArea()
{
    ...
}
```
Summary of Terminology (13)

- **Instance data field:**
  - A special variable declaration in the body of a class but outside any constructor or method.
  - Each object of the class has its own copies of the instance data fields.

- **Examples:**
  - `private double radius;`
  - `private double root1, root2;`
Summary of Terminology (14)

- **Constructor prototype:**
  - the first line of a constructor declaration

- **Examples:**
  - `public CircleCalculator(double r)`
  - `public TriangleCalculator(double a, double b, double g)`
Constructor call expression:
- An expression of the form
  \[ \textit{new ClassName(actualArguments)} \]
- purpose is to construct an object

Examples:
- \textit{new CircleCalculator(3.0)}
- \textit{new QuadraticRootFinder(1,2,-1)}
- \textit{new TriangleCalculator(1,1,90)}
Summary of Terminology (16)

- **Method prototype:**
  - the first line of a method declaration

- **Examples:**
  - `public void setC(double value)`
  - `private void doCalculations()`
  - `public double getArea()`
  - `public static double sqrt(double x)`

- Static means not associated with an object.
Method call expression:

- An expression of one of the forms
- `objectName.methodName(actualArguments)`
- `methodName(actualArguments)`

Examples:

- `circle1.getArea()`
- `doCalculations()`
- `triangle.setC(1.0);`

the object here is called "this"
Summary of Terminology (18)

- **Enquiry method:**
  - method that returns information specific to an object of a class, such as the value of an instance data field, without changing the object

- **Examples:**
  - getArea
  - getA
  - getAlpha
Summary of Terminology (19)

- **Mutator method:**
  - method that changes the state of an object of a class, usually by modifying one or more instance data fields

- **Examples:**
  - `setA`
  - `setB`
  - `doCalculations`
Get and set methods:
- enquiry methods are often called "get" methods
- mutator methods are often called "set" methods

Common convention:
- use `getName` to name a method that returns the value of an instance data field called name
- use `setName` to name a method that changes the value of an instance data field called name
Summary of Terminology (21)

- **Local variable:**
  - very different from an instance variable
  - declared in a constructor or method body
  - not available outside the method or constructor

- **Example:**
  - `double s` in TriangleCalculator constructor
  - `double d = ...` in doCalculations method
Summary of Terminology (22)

- **Formal argument:**
  - a special local variable declared in a constructor or a method prototype whose value is supplied when a method or constructor call expression is executed.

- **Examples:**
  - `public void setC(double value)`: value is a formal argument of type double
  - `public CircleCalculator(double r)`: r is a formal argument of type double
Summary of Terminology (23)

- **Actual argument:**
  - a variable or expression whose value is used as the value of the corresponding formal argument when a method or constructor call expression is executed (called)

- **Examples:**
  - `new CircleCalculator(3.0)`: 3 is the actual argument
  - `triangle.setC(3+x)`: the value of 3+x is the actual argument.
Summary of Terminology (24)

- **return statement:**
  - `return expression;`
  - `return;`

- The first form is used in a method that has a non-void return type and expression is the value of this type that is returned.

- The second form is used in a method that has a void return type.

- When a return statement is executed the method finishes execution immediately.