Introduction and Intermideate Java

By:
Shams Al Ajrawi

Computers and Java

Why Program?

- To use computers to perform many versatile functions
- Syntax - rules of programming languages that must be followed

Computer Systems: Hardware and Software

- Hardware - physical pieces of the computer

CPU	**Processor, Central Processing Unit**	**"brains"**	**Fetch, decode, execute**
RAM	Random Access Memory, Main Memory, Primary Memory	Volatile form of storage (contents are lost when power is lost)	**Stores portions of currently running programs and data**
Secondary Storage Devices	Hard drive (Magnetic, Solid State) USB Stick Floppy Disk CD	Non-volatile form of storage (contents are not lost when power is lost)	**Stores information for longer periods of time**
Input Devices	Keyboard, mouse, microphone, webcam, barcode reader		**Collect data from outside world**
Output Devices	**Monitors, speakers, printers, fax, projector**		**Send data to the outside world**

Software

- Software - Programs that run on a computer

Operating Systems (OS)	Windows, Mac OS, Linux, Android, iOS •types:	Manage computer's resources
Applications	Microsoft Word, Google Chrome, Safari, Firefox, Opera, Steam, PhotoShop, Photos, AutoDesk, Eclipse, Blender, Clash of Clans, Facebook, Maps, Discord	Specific Tasks

What is a Program Made Of?

- Programming Language, 5 elements:

1. Keywords	Reserved words in the Java language
2. Operators	Apply to values in expressions
3. Punctuation	Separate and enclose elements
4. Programmer-Defined Identifiers	Names that the programmer creates
5. Syntax	Rules for combining (1) - (4)

The Programming Process

- 1. Cleary define what the program is to do.
- 2. Visualize the program running on the computer.
- 3. Use design tools such as pseudo code to create a model of the program.
  - a. Display "How many hours did you work".
  - b. Input hours.
  - c. Display "How much do you get paid per hour?"
  - d. Input rate.
  - e. Store the value of hours times rate in the pay variable.
  - f. Display the value in the pay variable.
- 4. Check the model for logical errors (mistakes that cause the program to produce incorrect results)
- 5. Type the code, save it, and compile it.
- 6. Correct any error found during compilation. Repeat steps 5 and 6 as many times as necessary.
- 7. Run the program with test data for input.
-

Errors

Syntax	• Error in combining elements of the programming language • Cause program to fail to compile • Compile-time error
Logical	• Error in the calculations in the program • Won't prevent program from compiling, but will result in anomalous output
Run-time error	Errors that occur after the program is started • Logical Error • Other types of run-time errors

Object-Oriented Programming

- Classes - Categories
- Objects - Members of a class
  - a.k.a instances of a class
  - Creating an object based on a class: instantiate
  - An object is an instantiation of a class
  - Objects have:

    Data (things objects know) - a.k.a attributes

    e.g. title, author, publisher, page count, price

    e.g. speed, gas level, direction, size

    Methods (things they do) - a.k.a behaviors

    e.g. makeAvailable, checkOut, remove, add

    e.g. drive, brake, switchGear, switchToFourWheelDrive

    Encapsulation: Melding data and methods

PayCalculator.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package paycalculator;

import java.util.Scanner;

/**
 *
 * @author Alex Stiller
 */
public class PayCalculator
{

```

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args)
{
   //a. Display "How many hours did you work". System.out.println("How many hours did you work?");
   //b. Input hours.
   //set up the keyboard
   Scanner keyboard = new Scanner(System.in);
   //use nextInt() to read in the hours int hours = keyboard.nextInt();
   //c. Display "How much do you get paid per hour?" System.out.println("How much do you get paid per hour?");
   //d. Input rate.
   int rate = keyboard.nextInt();
   //double rate = keyboard.nextDouble();
  //e. Store the value of hours times rate in the pay variable. int pay = hours *
  rate;
  //f. Display the value in the pay variable. System.out.println("Hours: " + hours +
  ""
      + " Rate: $" + rate
      + " Gross pay is $" + pay);
  }
}
```

2. Java Fundamentals

- Parts of a Java Program
- The print and println Methods and the Java API
- Variables and Literals
- Primitive Data Types
- Arithmetic Operators
- Combined Assignment Operators
- Creating named constants with final
- The String Class
- Scope
- Comments
- Programming Style
- Reading Keyboard Input
- (Optional) Common Errors to Avoid
-

Parts of a Java Program

```
// This is a simple Java program.
public class Simple
{
    public static void main(String[] args)
      {
     System.out.println("Programming is great fun!");
       }
 }
```

- Every Java program needs a main method.
- Braces must be balanced.
- Statements are terminated with semicolons.
- Java is a case-sensitive language.
- All Java programs must be stored in a file with a .java extension.
- The filename must be the same as the class name.
- Comments are ignored by the compiler.
- A .java file may contain many classes but may only have one public class.
- If a .java file has a public class, the class must have the same name as the file.

The print and println methods and the Java API

- System.out.println("The cat sat"); System.out.println("on the mat");

Output:

```
The cat sat on the
Mat
```

- System.out.print("The cat sat"); System.out.print("on the mat");

Output:

```
The cat saton the mat
```

- The println method places a newline character at the end of whatever is being printed out.
- The print method does not place a newline character at the end of whatever is being printed out.

Java Escape Sequences

- System.out.print("The cat sat\n");
- System.out.print("on the mat");
- Output:

```
The cat sat
on the mat
```

\n	Advances the cursor to the next line
\t	Prints a tab
\b	Causes the cursor to back up one position
\r	Causes the cursor to move to beginning of the current line
\\	Prints a \
\'	Prints a '
\"	Prints a "

-
  ```
  System.out.print("These       are       our       top       sellers:\n");
  System.out.print("\tComputer                    games\n\tCoffee\n");
  System.out.print("\tAspirin");
  ```

-
  ```
  System.out.print("These are our top sellers:\n\tComputer games\n\tCoffee\n\tAspirin");
  ```

- Output:

```
These       are       our       top       sellers:
        Computer                games
        Coffee
        Aspirin
```

Variables and Literals

- **A variable is a named storage location in the computer's memory**
- **A literal is a value that is written into the code of a program**

```
double itemCost = 82.50;
double tr = 0.0725;
double salesTaxRate = 0.0725
```

- Java programs should be "self-documenting"

Primitive Data Types

- Variables have a name and a data type.
- Java has 8 primitive data types:

byte	Integers in the range -128 to 127
short	Integers in the range -32,768 to 32,767
int	Integers in the range -2,147,483,648 to 2,147,483,647
long	Integers in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point (decimal) numbers in the range of -3.4 x 10^38 to 3.4 x 10^38 7 digits of accuracy
double	Floating-point (decimal) numbers in the range of -1.7 x 10^308 to 1.7 x 10^308 15 digits of accuracy
boolean	true or false
char	characters

int counter = 0;

float percentage = 0.06; boolean isValid = true; char myCharacter = '1'; char myCharacter2 = 'a';

=	**Assignment Operator (assigns what is on right to what is on left)**
initialize	**Assign a value for the first time**

Ex.

double rate = 50; int hours = 40;

double pay = rate * hours;

Arithmetic Operators

- Java has 5 arithmetic operators:

+	**Addition**	**total = cost + tax;**
-	Subtraction	**cost = total - tax;**
*****	Multiplication	**tax = cost * rate;**
	Division	**salePrice = original / 2;**
%	**Modulus**	**remainder = value % 5;**

Combined Assignment Operators

x = x + 5; // adds 5 to x and stores result

// back in x

Operator	Example	Equivalent

+=	x += 5;
	x += z;
		x = x + 5; X = x + z;

-=	x -= 5;
	x -= y;
		x = x - 5;
		x = x - y;

*=	z *= 10;	z = z * 10;
/=	a /= b;	a = a / b;
%=	c %= 3;	c = c % 3;

Creating named constants with final

Constants - a special type of variable whose value cannot

change after it has been initialized

Constants are declared using keyword   final

Once initialized, constants cannot be changed.

By convention, constants are all upper case words separated by underscores

```
final double CAL_SALES_TAX = 0.0725;
```

The String Class

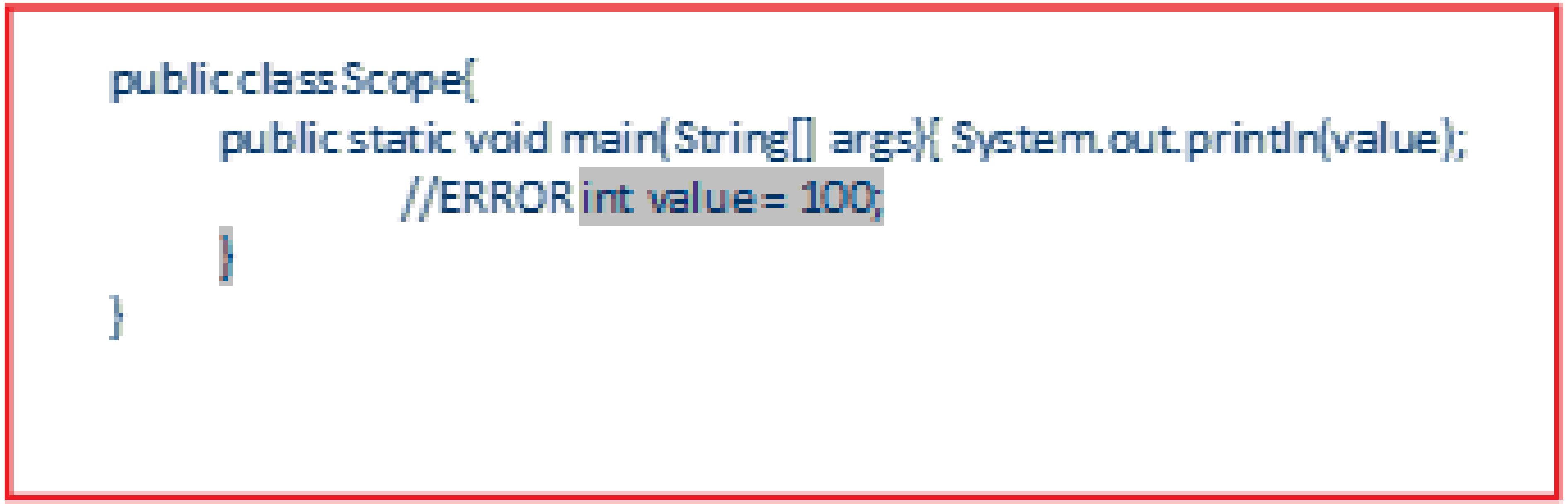
- Java has no primitive data type that holds a series of characters.
- The String class is used to store strings of text.
- Notice the S in String is upper case.
- Reference variable
  - Refers to an object in memory
  - Not a primitive data type, which are not represented as objects in memory
- A (reference) variable can be assigned a String literal
  - String value = "Hello";
    - OR
  - String value = new String("Hello");

```
public class StringDemo{
    public static void main(String[] args){
    String greeting = "Good morning, ";
    String name = "Herman";
        char closing = '!';
        System.out.println(greeting + name + closing);
    }
}
```

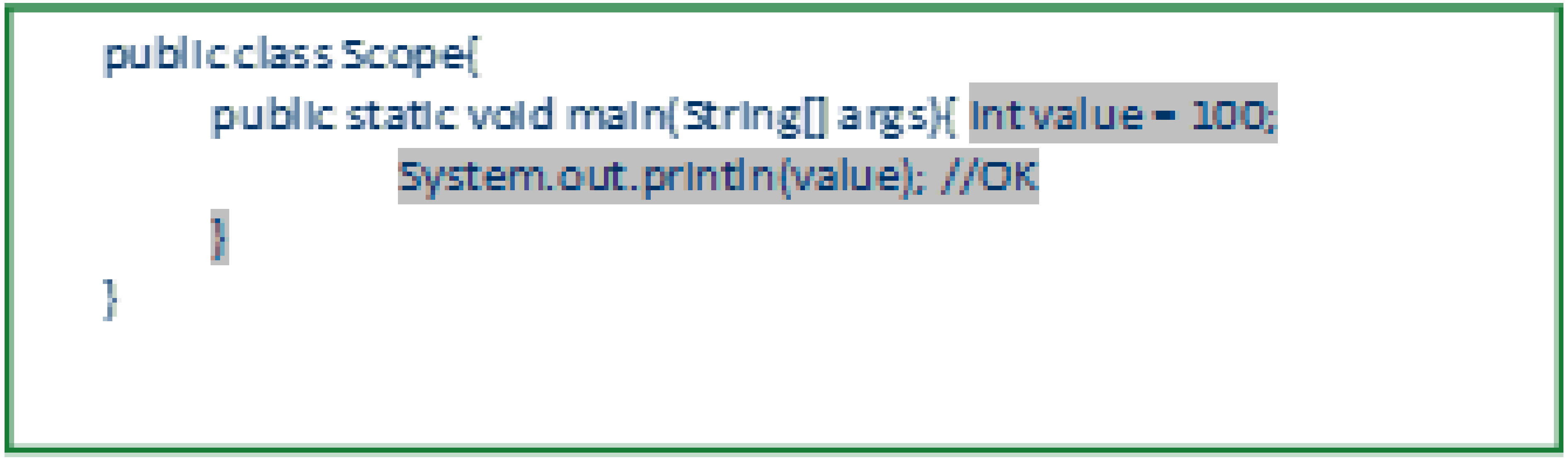
Output:

```
Good morning, Herman!
```

Scope

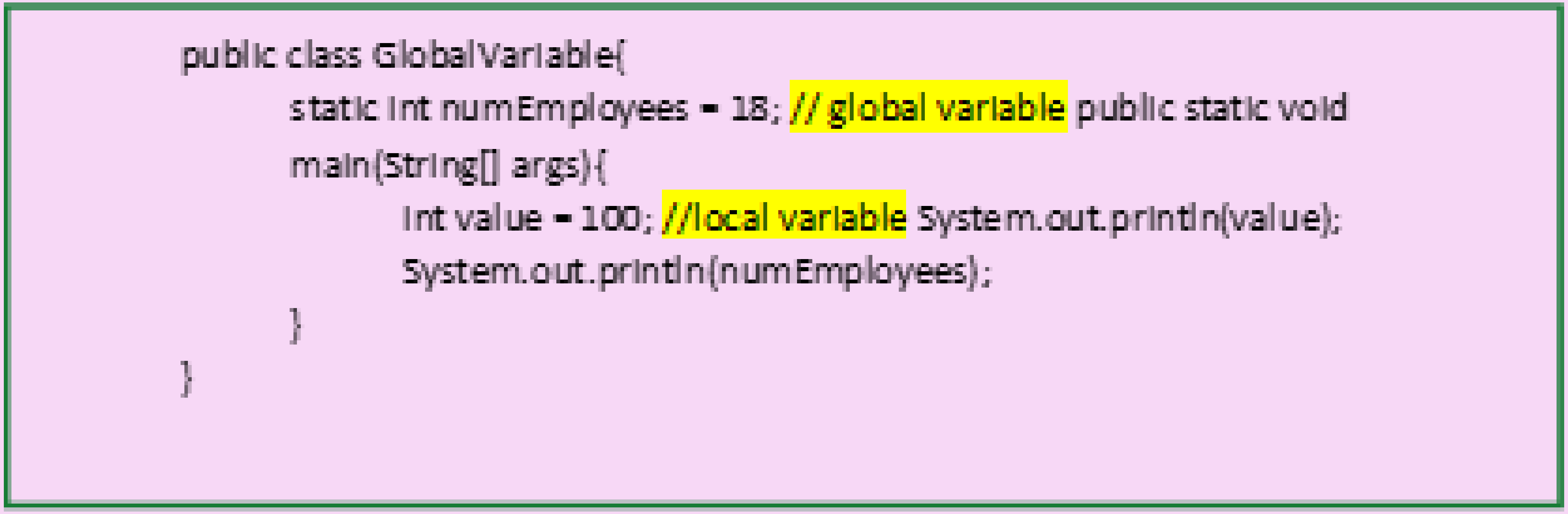


```
public class Scope{
        public static void main(String[] args){ System.out.println(value);
                //ERROR int value = 100;
        }
}
```



```
public class Scope{
        public static void main(String[] args){ int value = 100;
                System.out.println(value); //OK
        }
}
```

- Scope - the part of the program that has access to a variable
- local variables - variables declared inside a method (like the main method)
- global variables - variables declared outside a method
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.
- Global variables' scope begins at declaration of the variable and ends at the end of the class in which it was declared.



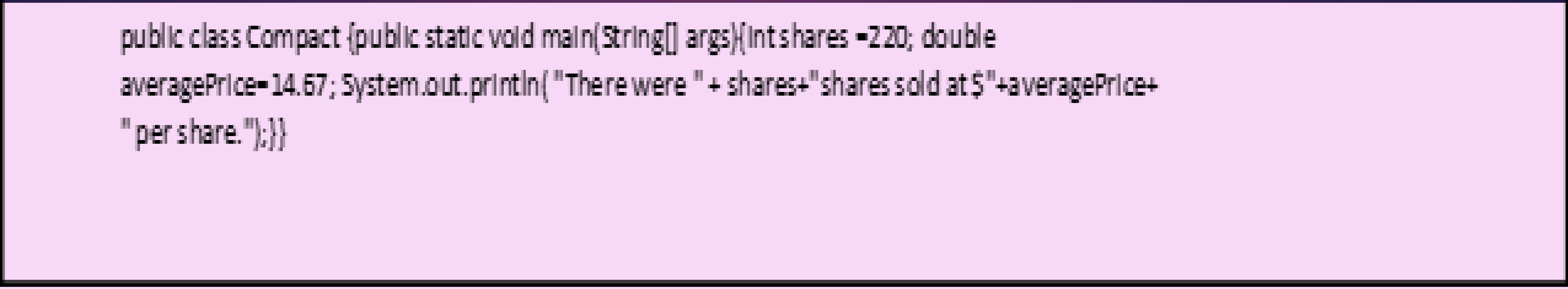
```
public class GlobalVariable{
        static int numEmployees = 18; // global variable public static void
        main(String[] args){
                int value = 100; //local variable System.out.println(value);
                System.out.println(numEmployees);
        }
}
```

Comments

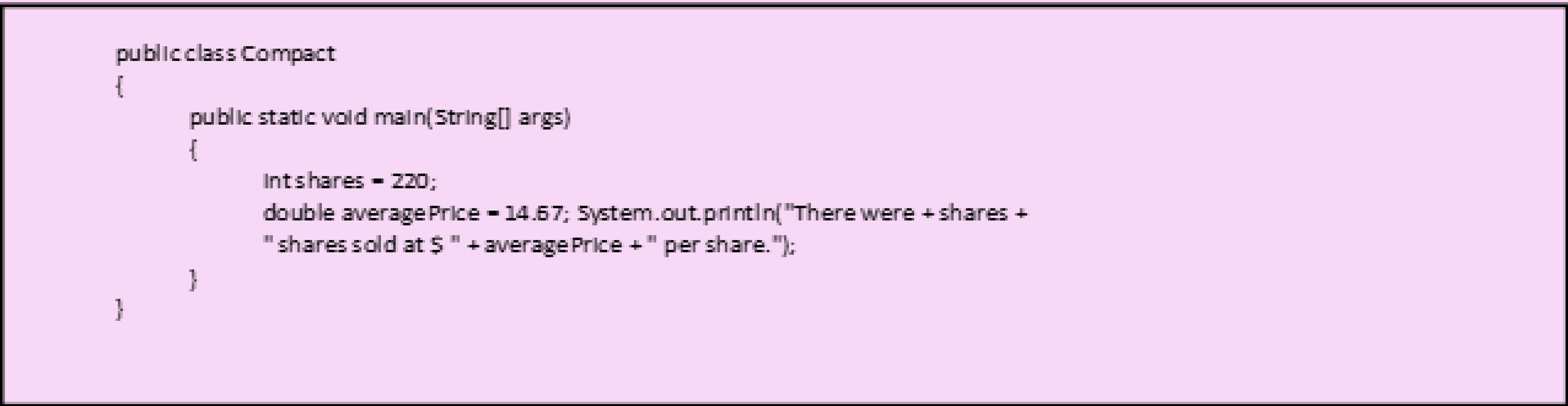
Comment Style	Description
//	Single line comment
/* … */	Block comment. Everything /* and */ is ignored.
/** … */	Javadoc comment

Programming Style

- Whitespace characters are all equivalent, Space, Tab, Newline, Carriage return



```
public class Compact {public static void main(String[] args){int shares =220; double
averagePrice=14.67; System.out.println( "There were " + shares+"shares sold at $"+averagePrice+
" per share.");}}
```



```
public class Compact
{
        public static void main(String[] args)
        {
                int shares = 220;
                double averagePrice = 14.67; System.out.println("There were + shares +
                " shares sold at $ " + averagePrice + " per share.");
        }
}
```

Reading Keyboard Input

- To read input from the keyboard, use the Scanner class.
- The Scanner class is defined java.util
- import java.util.Scanner;
- To create a Scanner object:

Scanner keyboard = new Scanner(System.in);

- System.in means make the Scanner object read input from the keyboard

Scanner methods:

Method	Example	Description

nextDouble	double number;
	Scanner keyboard = new Scanner(System.in);
	System.out.println("Enter a value: ");
	 number = keyboard.nextDouble();
		Return input as a double.

nextInt	int val;
	
	Scanner keyboard = new Scanner(System.in);
		Return input as an
		
		int.

	System.out.println("Enter a value: ");
	 number = keyboard.nextInt();	

nextLine	String name;
	Scanner keyboard = new Scanner(System.in);
	System.out.println("Enter your name: ");
	name = keyboard.nextLine();
		Return input as a String.

```
import java.util.Scanner;

public class InputExample{
	public static void main(String[] args){
		String name; int age; double income;

		Scanner keyboard = new Scanner(System.in); System.out.println("What is your name? "); name = keyboard.nextLine();

		System.out.println("What is you age? "); age = keyboard.nextInt();

		System.out.println("What is your annual income? "); income = keyboard.nextDouble();

		//Display the information back to the user. System.out.println("Hello, " + name + ". \nYour age is " +
						age + " and your income is $" + income);
	}

}
```

3. Decision Structures

- The if Statement
- The if-else Statement
- Nested if Statements
- The if-else-if Statement
- Logical Operators
- Comparing String Objects
- The Conditional Operator
- The switch Statement
- The System.out.printf method

The if Statement

- Causes a statement or group of statements to run if a condition is true.
- The statement or group of statements is simply skipped if the condition is false.
- General Format:

```
if (condition)
     {
            statement1;
           statement2;
            ….
     }
```

- If there is only one statement, you can drop the curly braces:
- if (condition)

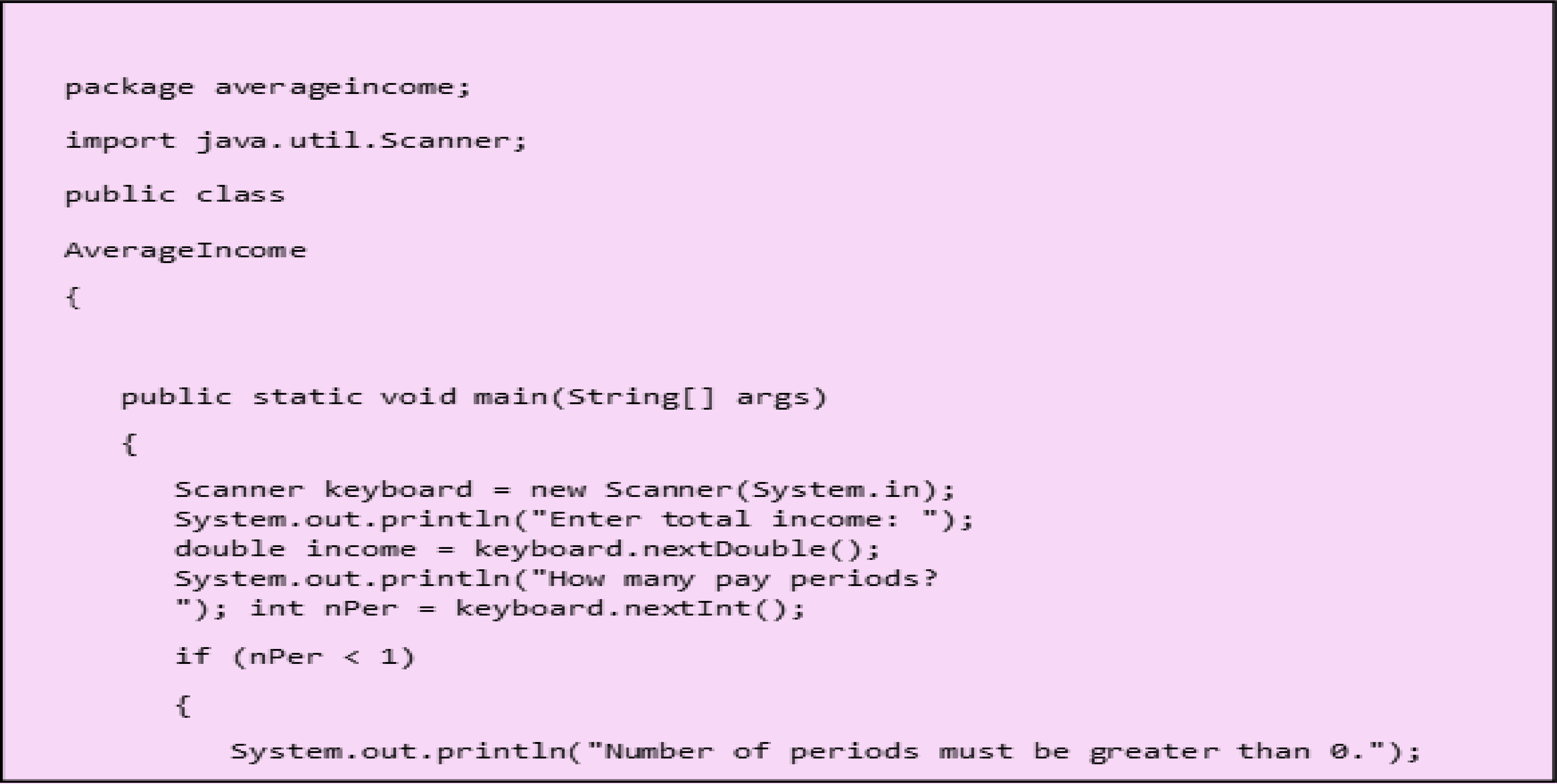
```
            statement1;
```

```
package mysterynumber;

//This program demonstrates if statements
import java.util.Scanner;
public class SecretNumber{
    public static void main(String[] args){ int input; //To hold the user'sinput
            //Create a Scanner object for keyboard input
            Scanner keyboard = new Scanner(System.in);

            //Prompt the user to enter a guess
            System.out.print("Enter a guess whole number between 1 and 10:"); input = keyboard.nextInt(); //read the input,
            store to input

            if (input > 10){
                 System.out.println("Your guess is too big. Better luck next time.");
            }
                f (      t < 1) {
                 System.out.println("Your guess is to small. Better luck next time.");
            }
            //Determine whetyher the user entered the mysterynumber. if (input == 9){
                 System.out.println("Congratulations! You know the mystery number!");
            }
           else{
                System.out.println("Sorry, that is NOT the mystery number!");
        }

    }
}
```

Relational Operators

>	Is Greater Than
<	Is Less than
>=	Is Greater Than Or Equal To
<=	Is Less Than Or Equal To
==	Is Equal To
!=	Is Not Equal To

The if-else Statement



```
package averageincome;

import java.util.Scanner;

public class

AverageIncome

{



    public static void main(String[] args)

    {

        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter total income: ");
        double income = keyboard.nextDouble();
        System.out.println("How many pay periods?
        "); int nPer = keyboard.nextInt();

        if (nPer < 1)

        {

            System.out.println("Number of periods must be greater than 0.");
```

The if-else-if Statement

```
import java.util.Scanner;
public class GradeCalculator{
    public static void main(String[] args){ int grade;
          Scanner input = new Scanner(System.in); System.out.println("Enter a numeric grade out of
          100: "); grade = input.nextInt();

          if (grade >= 90){
               System.out.println("Congratulations! You got an A!");
          }
        else if     (grade >= 80){
               System.out.println("Yay! You got a B!");
          }
        else if     (grade >= 70){
               System.out.println("You got a C!");
          }
        else if     (grade  >=  60){ System.out.println("Oh no. You got a D!");
          }
        else     { //trailing else System.out.println("Sorry. You got an F.");
          }
    }
```

Logical Operators

Short Circuiting

&&	false as soon as one of its operands is false
||	true as soon as one of its operands is true

```
boolean bool1 = true;
boolean bool2 =false;
```

$$\underbrace{!\underset{T}{\text{bool1}}}_{F} \;\&\&\; \underset{F}{\text{bool2}}$$

false

The Conditional Operator

```
Condition ? Value1: Value2

if (hours < 5) {
        hoursToBill = 5;
}
else {
        hoursToBill = hours;


hoursToBill = hours < 5 ? 5 : hours;

------------------------------------------


String message = "";
int grade = keyboard.nextInt(); if (grade >
90) {
          message = "Honors";
}
else {
        message = "NA";
}
```

message = grade > 90 ? "Honors" : "NA";

The switch Statement

- Worked like if-else-if

```
switch (SwitchExpression){ case
    CaseExpression:

            statement1;

            statement2;

            …

            break;

        case CaseExpression:

            statement1;
            statement2;

            …

            break;
    }
```

4. Loops

- The Increment and Decrement Operators
- The while Loop
- Using the while Loop for Input Validation
- The do-while Loop
- The for Loop
- Running Totals and Sentinel Values
- Nested Loops
- The break and continue Statements
- Deciding Which Loop to Use
- Generating Random Numbers with the Random Class
-

The Increment and Decrement Operators

Increment Operator	++
Decrement Operator	--

The while Loop

- while (condition) {
- statements;
- }

- • While the condition is true, the loop iterates.
- • Pre-test loop - condition is checked prior to executing the loop.
- • Infinite loop - A loop that never terminates

The do-while Loop

```
do {
        statements;
    }
    while (condition);
```

```
package loops;

import java.util.Scanner;

public class Loops
{

    public static void main(String[] args)
    {
        //set up a Scanner object
        Scanner input = new Scanner(System.in);
        int number = 50;
        do
        {
            if (number > 100 || number < 0)
            {
                System.out.println("Invalid input");
            }
            System.out.println("Enter a number between 1 and 100");
            number = input.nextInt();

        } while (number > 100 || number < 0); System.out.println("You

        entered " + number);

    }
}
```

The for loop

```
for (int i = 0; i < 100; i++) {
        statements;
}
```

```java
package loops;

import java.util.Scanner;

public class RunningTotal
{

   public static void main(String[] args)
   {
      double total = 0; //accumulator variable
      Scanner input = new Scanner(System.in);
      for (int day = 1; day <= 5; day++)
      {
         System.out.println("Enter sales for day " + day);
      total += input.nextDouble(); //increments total by daily sale amount
      }
      System.out.printf("Total: $%.2f\n", total);
   }
}

```

Nested Loops

- Nested loop - one loop inside another
- Inner and Outer loop
- Inner loop will execute all of its iterations for every one execution of the outer loop

The break and continue Statements

- break Statement
  - Terminates a loop
- continue Statement
  - Causes the currently executing iteration of a loop to terminate

    Then the next iteration begins

Deciding which Loop to Use

Use where there is some type of counting variable that can be evaluated	for
Use where you want the statements to execute at least one time	do while
Use where you do not want the statements to execute if the condition is false in the beginning	while

Methods

- Introduction to Methods
- Passing Arguments to a Method
- More about Local Variables
- Returning a Value from a Method
- Problem Solving with Methods

Method - bundle of statements that performs a particular task

types of methods:

- Void methods
  - Perform a task then terminate
    - `System.out.println("Hi");`
- Value returning methods
  - Perform a task then send a value back to the rest of the program
    - `int x = Integer.parseInt("100);`
    - `double y = Math.pow(2, 3);`

Why Write Methods?

- Functional Decomposition - (Divide and Conquer), breaks a problem down into small, manageable pieces.
- Reuse - Methods can be used from several locations in a program

Passing Arguments to a Method

Argument - Value that is passed into a method when it is called

- The data type of an argument in a method call must correspond to the variable declaration in the parentheses of the method definition

Java will automatically perform widening conversions

(int where a double is expected) but narrowing conversions (double where an int is expected) will cause a compiler error

```
public static void showSum(double num1, double num2)
        { double result = num1 + num2;
        System.out.println("The sum is " + result);
}
```

```
showGreeting(18, "Justin");
```

```
public static void showGreeting(int age, String name)
        { System.out.println("Hi, Your name is " + name);
        System.out.println("Your age is " + age);
}
```

- Pass-by-Value - Copy of argument is made. The copy is sent to the method. The method cannot change the original value.
    - All arguments of the primitive data types are passed by value

- Pass-by-Reference - Memory address of the argument is sent to the method. The method can change the original value.
    - All arguments that are reference types (i.e. objects, i.e not primitive data types) are passed by reference.

```
/*
 To change this license header, choose License Headers in Project
   Properties.
 To change this template file, choose Tools | Templates
 and open the template in the editor.
 */
package passbyvalue;

/**
 *
 @author student
 */
public class PassByValue
{

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args)
   {
     int x = 4;
     String z =  "Stiller"; Team
     yankees = new Team();
     //before

     System.out.println("x is " + x);
     System.out.println("z is " + z);
     System.out.println("wins is " + yankees.wins);

     System.out.println(timesTen(x));
     System.out.println(addA(z));
     System.out.println(addWin(yankees));

     //after
     System.out.println("x is " + x);
     System.out.println("z is " + z);
     System.out.println("wins is " + yankees.wins);
```

```
}

public static int timesTen(int x){
x = x * 10;
  return x;
}

public static String addA(String b){

  return x;
}

public static String addA(String b){
b = b + "a";
return b;
}

public static Team addWin(Team c)
{
c.wins = c.wins + 1;
return c;
}
```

Output:

```
x is 4
z is Stiller
wins is 0
40
Stillera
passbyvalue.Team@15db9742
 x is 4
z is Stiller
wins is 1
```

More about Local Variables

- A local variable is declared inside a method
- Local variables only have scope from line in which

they are declared to the end of the method in which they are declared.

- Different methods can have local variables with the same name.
- When the method ends, the local variables (including the parameter variables) are destroyed and any values stored are lost.
- Local variables are not automatically initialized with a default value and must be given a value before they can be used.

Returning a Value from a Method

```
/**
*This method adds two numbers
*@param num1 our first number to add
*@param num2 our second number to add
*@return returns the sum of the two integer values
*/
public static int sum(int num1, int num2){ return num1 + num2;
}
/*
This method displays the full name.
@param firstName the first name
@param lastName the last name
@return the full name
*/
public static String fullName(String firstName, String lastName){ return firstName + " " + lastName;
}
```

Problem Solving with Methods

- A large, complex problem can be solved a piece at a time using methods.
- The process of breaking a problem down into smaller pieces is called 'functional decomposition'
- If a method calls another method that has a throws clause in its header, then
- the calling method should have the same throws clause.
- All methods that use a Scanner object to open a file must throw (or handle) IOException

A First Look at Classes

- Objects and Classes
- Writing a Simple Class, Step by Step
- Instance Fields and Methods
- Constructors
- Passing Objects as Arguments
- Overloading Methods and Constructors
- Scope of Instance Fields
- Packages and import Statements
- Focus on Object-Oriented Design: Finding the Classes and Their Responsibilities

Objects and Classes

- Objects have two general capabilities:
  - Store data (fields)
  - Perform operations (methods)
- -Classes
  - Blueprint/Template for objects
- Instance of a class - Object created from a class

- Java API (Application Programming Interface) provides many pre-written classes

e.g. Scanner, String, File, PrintWriter, JOptionPane

Writing a Simple Class, Step by Step

Constructor - method that is used to perform operations at the time an object is created

-typically initializes instance fields

Class Member - Field or Method

Access Specifiers - public - the member can be accessed

inside the class or outside

private - the member can be accessed inside the

class but cannot be accessed outside the class

Data Hiding - Declaring instance fields as private, only accessing fields through methods

```
/*
*To change this license header, choose License Headers in Project Properties.
*To change this template file, choose Tools | Templates
*and open the template in the editor.
*/
package worldseries;


/**
*
*@author student
*/
public class Team
{
//These are the fields.
private int wins;
 private int losses;
 private String name;
//private double winningPercentage;

//This is a constructor.
public Team(String n, int w, int l){
name = n; wins = w; losses = l;
}

public String getName(){
return name;
}

public int getWins(){ return wins;
}

public int getLosses(){
return losses;
```

```
public double getWinningPercentage(){ double total = wins + losses; return wins / total;
}

public void printStats(){ System.out.println("Name: " + name); System.out.println("Wins: " + wins);
System.out.println("Losses: " + losses);
System.out.println("WP: " + getWinningPercentage());


}

public void setWins(int w){
wins = w;
}

public void setLosses(int l){
losses = l;
}
```

WorldSeries.java

```
/*

*       To change this license header, choose License Headers in Project Properties.

*       To change this template file, choose Tools | Templates

*       and open the template in the editor.

*/

package worldseries;

/**

*       @author student

*/

public class WorldSeries

{

/**

* @param args the command line arguments

*/

public static void main(String[] args)

{

Team cubs = new Team("Cubs", 92, 70);

Team nationals = new Team("Nationals", 97, 65);

Team indians = new Team("Indians", 102, 60);
```

```
//Starting statistics
cubs.printStats();
 nationals.printStats();
indians.printStats();
indians.printStats();

System.out.println("Now entering the playoffs...");

//cubs win, nationals lose, indians lose

//cubs win cubs.setWins(cubs.getWins() + 1);

//nationals lose
nationals.setLosses(nationals.getLosses() + 1);

//indians lose
indians.setLosses(indians.getLosses() + 1);
//Ending statistics

cubs.printStats();
nationals.printStats(); indians.printStats();
}
}
```

Instance Fields and Methods

- instance fields - fields that belong to an object of the class
- instance methods - methods that can be called on an object of the class
- wins, losses, winningPercentage are instance fields. They belong to a particular instance of the Team class (i.e. cubs).

Constructors

- Constructor - Method used to perform operations on a newly created object
- If you do not write a constuctor, Java provides one when the class is compiled. This is known as the default constructor
- Default Constructor

Numeric Fields	**0**
boolean	**false**
Reference Variables	**null**

Team.Java

```
/*
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
*/
package worldseries;


/**
*
@author student
*/
public class Team
{
//These are the fields. private int wins; private int losses; private String name;
//private double winningPercentage;


//This is a constructor.
/*public Team(String n, int w, int l){
name = n; wins = w; losses = l;
}
*/
```

```
public String getName(){
return name;
}

public int getWins(){
return wins;
}

public int getLosses(){
return losses;
}

public double getWinningPercentage(){ double total = wins + losses; return wins
/ total;
}

public void printStats(){ System.out.println("Name: " + name);
System.out.println("Wins: " + wins); System.out.println("Losses: " + losses);
System.out.println("WP: " + getWinningPercentage());

}

public void setWins(int w){
wins = w;
}

public void setLosses(int l){ losses = l;
}
}
```

WorldSeries.java

```
/*
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
package worldseries;

/**
*
        @author student
*/
public class WorldSeries
{

/**
* @param args the command line arguments
*/
public static void main(String[] args) {
/*Team cubs = new Team("Cubs", 92, 70);
Team nationals = new Team("Nationals", 97, 65);
Team indians = new Team("Indians", 102, 60);
Team cubs = new Team();
Team nationals = new Team();
Team indians = new Team();
//Starting statistics
cubs.printStats(); nationals.printStats(); indians.printStats();
System.out.println("Now entering the playoffs...");

//cubs win, nationals lose, indians lose
//cubs win
cubs.setWins(cubs.getWins() + 1);
//nationals lose nationals.setLosses(nationals.getLosses() + 1);

//indians lose
indians.setLosses(indians.getLosses() + 1);


//Ending statistics cubs.printStats();
nationals.printStats(); indians.printStats();
}
}
```

- Default Constructor has no parameters. You can also write a no-arg constructor
- //This is a constructor.

```
public Team(String n, int w, int l){
name = n;
 wins = w;
 losses = l;
}
//This is a no-arg constructor.
public Team(){
name = "None Given";
}
```

Overloading Methods and Constructors

- Method Overloading - Two or more methods (including constructors) in a class can have the same name as long as their parameter lists are different

```
public int add(int num1, intnum2){ int result = num1 + num2; return
          result;
}

public int add(int num1, int num2, int num3){ int result = num1 + num2 + num3; return
          result;
}

public String add(String str1, String str2){ String combined = str1 + str2;
          return combined;
```

Method binding - process of matching a method call with the correct method. The compiler uses the data types and number of parameters in the parameter list

Method signature - name and parameter list

Instance Method - a non-static method

Packages and import Statements

- • Classes are typically grouped into packages.
- • Example:

  java.util contains the Scanner class
- Explicit and Wildcard import statements
- import java.util.Scanner; // allows Scanner class to be used
- import java.util.*; //allows all classes in java.util package to be used
- The java.lang package is the only package automatically made available to any Java class. System class is part of java.lang package.

java.applet	Provides class to create an applet
java.awt	Provides classes for creating GUIs
java.io	Provides classes that perform input, output, and file operations
java.lang	Provides general classes for the Java language (automatically imported)
java.net	Provides classes for network communications
java.security	Provides classes that implement security features
java.sql	Provides classes for using databases.
java.text	Provides classes for formatting text
java.util	Provides various utility classes.
javax.swing	Provides classes for creating GUIs

Focus on Object-Oriented Design: Finding the Classes and Their Responsibilities

- Finding the classes

  Get written description of the problem domain Identify all nouns, each is a potential class

  Refine list to include only classes relevant to the problem

- Identify the responsibilities

  Things a class is responsible for knowing Things a class is responsible for doing

Case Study

- ***: Minimum requirement to qualify for grading: 1) Code must be properly indented, modularized and follows the best practices.***

  Account Class (A bank account class with methods to deposit, withdraw, and check the balance.) (5 points)
  1. Class contains 3 private data members called accountBalance, accountName and accountNum. Choose appropriate data types for these variables
  2. Write an overloaded constructor that takes one parameter of the type string. The constructor initializes the accountName with the parameter specified. The member variables accountBalance and accountNum are assigned some random value (No need to use rand function).
  3. Write appropriate destructor, set, get and display functions.
  4. The class will also contain member functions named menu, withdraw, and deposit

  The withdraw function checks to see if balance is sufficient for withdrawal. If so, decrements balance by amount; if not, prints an appropriate message indicating insufficient funds and displays the current balance.

  The deposit function adds deposit amount to balance and displays new balance

  The menu function will contain the following menu options: 1. Check Balance, 2. Deposit Amount, 3. Withdraw Amount, 4. Exit. Use appropriate messages for each option selected and then call the correct function to perform the task

Output:

```
Welcome to the Bank Account program
Here is your Initial Account Information:
Account for: Helen
Account #: 123456
Balance: $187.5

***********************
Menu
***********************
1. Check Balance
2. Deposit Amount
3. Withdraw Amount
4. Exit

Please make a selection: 8

ERROR: Invalid choice. Please try again !!!

***********************
Menu
***********************

1. Check Balance
2. Deposit Amount
3. Withdraw Amount
4. Exit

Please make a selection: 1

Here is the account Information:
Account for: Helen
Account #: 123456
Balance: $187.5

***********************
Menu
***********************
1. Check Balance
2. Deposit Amount
3. Withdraw Amount
4. Exit

Please make a selection: 2
Enter the amount you want to deposit: 50
Depositing $50 to Acc# 123456
Your new balance is now $237.5

***********************
Menu
***********************
1. Check Balance
2. Deposit Amount
3. Withdraw Amount
4. Exit

Please make a selection: 3

Enter the amount you want to withdraw: 500
ERROR: Cannot withdraw amount due to Insufficient Funds !
Your balance is now $237.5
```

```java
import java.util.Scanner;

public class Bank_Account {
	private int AccountNum;
	private double AccountBalance;
	private String AccountName;

	public Bank_Account(String AccountName, int AccountNum, double AccountBalance)

	{
		setAccountName(AccountName);
		setAccountBalance(AccountBalance);
		setAccountNum(AccountNum);
	}

	public void setAccountName(String AccountName) {
		this.AccountName = AccountName;
	}

	public String getAccountName() {
		return AccountName;
	}

	public void setAccountNum(int AccountNum) {
		this.AccountNum = AccountNum;
	}


	public int getAccountNum() {
		return AccountNum;
	}

	public void setAccountBalance(double accountBalance) {
		this.AccountBalance = accountBalance;
	}

	public double getAccountBalance() {
		return AccountBalance;
	}

	public void deposit(int Amount) {

		setAccountBalance(getAccountBalance() + Amount);
		System.out.println("");
		checkBalance();
	}
```

```java
private void display() {
          System.out.println("Name     : " + AccountName);
          System.out.println("Number   : " + AccountNum);
          System.out.println("Balance : " + AccountBalance);



}


public class ATM {

     public static void main(String[] args) {

          Bank_Account obj1 = new Bank_Account("Helen", 123456, 1187.5);
          obj1.Menu();

     }

}
```

```java
public void wihtdraw(int Amount) {


        if (Amount > getAccountBalance()) {
                System.out.println("insufficient funds ");
                System.out.printf("your current balance is: ", getAccountBalance());
        } else
                setAccountBalance(getAccountBalance() - Amount);
        checkBalance();
}

public void checkBalance() {
        System.out.printf("Current Balance is $%,.2f", getAccountBalance());
        System.out.println("");
}

public void Menu() {

        int input;
        do
        {
                System.out.println("Welcome to the Bank Account program");

                System.out.println("*****************************************");
                System.out.println("1- Check Balance");
                System.out.println("2- Deposit Amount");
                System.out.println("3- Withdraw Amount");
                System.out.println("4- Exit");

                display();

                System.out.println("Make your selection : ");
                Scanner scan1=new Scanner(System.in);
                input = scan1.nextInt();
        switch(input)
        {

                case 1:
                        checkBalance();
                        break;
                case 2:
                        System.out.println("Enter your Amount: ");
                        Scanner scan3=new Scanner(System.in);
                        int Amount = scan3.nextInt();
                        deposit(Amount);
                        break;
                case 3:
                        System.out.println("Enter your Amount: ");
                        Scanner scan2=new Scanner(System.in);
                        int Amount1 = scan2.nextInt();
                        wihtdraw(Amount1);
                        break;
                case 4:
                        System.out.println("Bye Bye ");
                        break;
                default:
                        System.out.println("Invalid Choice");
                }
        } while (input != 4);

}
```

Arrays

- Introduction to Arrays
- Processing Array Elements
- Passing Arrays as Arguments to Methods
- Some Useful Array Algorithms and Operations
- Returning Arrays from Methods
- Two-Dimensional Arrays
- The Selection Sort and the Binary Search Algorithms
- The ArrayList Class

Introduction to Arrays

Data Structures - Collections of related data items

Arrays – Data structure consisting of related data items of the same type

Remain the same length once they are created

Created with the keyword new

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

```
//Declare an array of 12 int elements
int[] x = new int[12];
```

data type name Size Declarator (must be an int > 0)

```
int[] c;
c = new int[12];
```

Numeric primitive-type elements	0
Boolean	false
String or any reference type	null

```
c[0] = 31;
c[1] = 28;
c[2] = 31;
c[3] = 30;
c[4] = 31;
c[5] = 30;
c[6] = 31;
c[7] = 31;
c[8] = 30;
c[9] = 31;
c[10] = 30;
```

initialization Lists

```
int x = 31; int y = 30;
int[] d = {x, 28, 31, y, 31,
30, 31, 31, 30, 31, 30, 31};
```

Processing Array Elements

- Array – Group of elements of the same type
- Arrays are objects (reference type)
- Elements in the array can be either primitive types or reference types

To refer to a particular element in an array, use the element's

 index scores[2] = 85.8; // reassign value in array

System.out.println(scores[3]); //prints 95.1

```
public static void main(String[] args)  {
        int[] numbers =
        {3, 6, 9};
int[] daysInMonths = new int[12];
daysInMonths[0] = 31;
daysInMonths[1] = 28;
daysInMonths[2] = 31;
daysInMonths[3] = 30;
daysInMonths[4] = 31;
daysInMonths[5] = 30;
daysInMonths[6] = 31;
daysInMonths[7] = 31;
daysInMonths[8] = 30;
daysInMonths[9] = 31;
daysInMonths[10] = 30;
daysInMonths[11] = 31;
System.out.println("This is a regular For Loop");
for (int x = 0; x < numbers.length; x++) 39    {
        System.out.println(numbers[x]);   }
//enhanced for loop
System.out.println("Behold: Enhanced For Loop");
for (int val : numbers){
System.out.println(val);
        }
System.out.println("This is a regular For Loop");
for (int x = 0; x < daysInMonths.length; x++)       {
System.out.println(daysInMonths[x]);  }
//enhanced for loop
System.out.println("Behold: Enhanced For Loop");
for (int val : daysInMonths){
System.out.println(val); }
        }
}
}
```

String Arrays

- Arrays of String objects

```
for (String s : employees){
System.out.println(s.length());
}
```

The ArrayList Class

Similar to arrays

ArrayList objects allow for Dynamic resizing: Accomodating more or fewer elements at execution time.

Reduces application development time.

import java.util.ArrayList;

Required for ArrayList class

ArrayList is an example of a Generic Class - Class that works with different types of Objects

- ArrayList<String>
- ArrayList<Employee>
-
- ArrayList<int> // BAD
- ArrayList<Integer> // GOOD
-

Methods of the ArrayList Class

add	Adds an element to the end of the ArrayList
add	Inserts an element at a given position in the ArrayList
clear	Removes all elements from the ArrayList
contains	Returns true if specific element is in the ArrayList
get	Returns element at specified index
indexOf	Returns index of the first occurrence of a specified element
trimToSize	Shrinks the capacity of the ArrayList to the current number of elements
size	Returns the number of elements in the ArrayList

```
package arraylists;

import java.util.ArrayList; public class ArrayListDemo{

public static void main(String[] args){

//Create an array list of Strings

ArrayList<String> items = new ArrayList<String>(); //
constructor


//Add red

items.add("red");


//Insert yellow at index 0

items.add(0, "yellow");


//Add one more

items.add("brown");


//Print out size

System.out.println("Size: " + items.size());


for (int i = 0; i < items.size(); i++){
System.out.println(items.get(i));

}

}

}
```

Inheritance

- What is inheritance?
- Calling the Superclass Constructor
- Overriding Superclass Methods
- Protected Members
- Chains of Inheritance
- The Object Class
- Polymorphism
- Abstract classes and Abstract Methods
- Interfaces

What is Inheritance?

- Inheritance - the ability to create new classes based on members (fields and methods) of existing classes

  Beetle is an insect Truck

  is an automobile

- Is-a relationship

- Superclass/ Parent / Base Class Subclass / Child Class / Derived Class

- Subclass inherits all non-private methods and data of the superclass.

- extends keywords
-
- Employee Example
- CommissionEmployee
- BasePlusCommissionEmployee

- BasePlusCommissionEmployee is a ComissonEmployee
- CommissionEmployee is the superclass.
- BasePlusCommissionEmployee is the subclass.
-

CommissionEmployee.java, BasePlusCommissionEmployee.java, and EmployeeTest.jav

```
public class CommissionEmployee
{

 //private instance variables private double commissionRate;
 private double grossSales;

    public CommissionEmployee(double c, double g)

    { commissionRate = c;
    grossSales = g;

 }
 public double getCommissionRate(){
  return commissionRate;
 }
return commissionRate;                                    }
}
public double getGrossSales(){
return grossSales;
}

public void setCommissionRate(double c){
commissionRate = c;
}

public void setGrossSales(double g){
grossSales = g;
}

public double getEarnings(){
return grossSales * commissionRate;
}
```

```
public class BasePlusCommissionEmployee extends
CommissionEmployee

{

//private instance fields

 private double commissionRate; private double
grossSales; private double baseSalary;


public BasePlusCommissionEmployee(double c,
double g, double b){


//comissionRate = c;

//grossSales = g; super(c, g);


baseSalary = b;


}

public double getBaseSalary(){

return baseSalary;

}


public void setBaseSalary(double b){

baseSalary = b;

}


@Override //  Optional public double getEarnings(){

return baseSalary + super.getEarnings();

//return baseSalary + commissionRate * grossSales;

//return commissionRate * grossSales;

}



}
```

```
public class EmployeeTest
{
public static void main(String[] args){


BasePlusCommissionEmployee bpEmployee = new BasePlusCommissionEmployee(0.1, 500, 2500);
CommissionEmployee cEmployee= new CommissionEmployee(.2, 1000);
```

```
 BasePlusCommissionEmployee bpEmployee = new.
 BasePlusCommissionEmployee(0.1, 500, 2500)
 ;
 CommissionEmployee cEmployee= new CommissionEmployee(.2, 1000);

 System.out.println("Earnings Report: ");
 System.out.println("Employee 1: " + bpEmployee.getEarnings());
 System.out.println("Employee 2: " + cEmployee.getEarnings());
 }
 }
```

Constructors in Subclases

- Use super() to pass in arguments to the superclass
- Call to super() must be first if it appears in the subclass constructor
-

Overriding Methods in Subclasses

- Overriding Methods - Subclass defines a method with the same name and parameter list as a method in its

- If a subclass overrides a method in the superclass, the subclass version
- will run unless super.methodName() is called
- E.g. super.getEarnings()

Protected Members

- Protected Members - methods or fields that are available in the same class, package, and subclass in which they are

Access Level Modifier	Class	Package	Subclass	World
No modifier	Y	Y	N	N
public	Y	Y	Y	Y
protected	Y	Y	Y	N
private	Y	N	N	N

Chains of Inheritance

* Java does not support multiple inheritance

* But, 1 class can extend another, which in turn extends another, and so on.

* Chain of Inheritance

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            javax.swing.JComponent
                javax.swing.JOptionPane
```

Polymorphism

- Polymorphism - aiming a superclass reference at a subclass object Animal a = new Animal();
- Beetle b = new Beetle();
- Animal a = new Beetle(); // polymorphism

```
import java.util.ArrayList;
public class ObjectDemo
{
public static void main(String[] args)
{
		Object year = new Integer(2);
		Object name = new String("Lonzo");
		Object fg = new Double(.67);
ArrayList<Object> list = new ArrayList<Object>();

list.add(year); list.add(name); list.add(fg);

for (Object obj : list)
{
System.out.println("List contains: " + obj);
}
```

Abstract Classes and Abstract Methods

- Abstract Class - class that cannot be instatiated
  - Intended to provide a superclass from which other classes can inherit
  - Other classes can share a common design based off the abstract superclass
-
- public class ClassName
- public abstract class ClassName
- Abstract Method - method in an abstract class
  - All methods in an abstract class are abstract methods.
  - Abstract methods
    - Have no body
    - Must be overridden in a subclass
- public double calculateArea(int x, int y){
  return x * y;

}
- public abstract calculateArea(int x, int y);
- * If a subclass does not override an abstract method, a compiler error will result.
- Any class that contains an abstract method is automatically abstract.

```
public abstract class Shape{
  public abstract double calculateArea();
}
```

```
public class Triangle extends Shape{ private double
  base, height;

  public Triangle(double b, double h){ base = b;
    height = h;
  }

  @Override
  public double calculateArea(){
    return base * height / 2.0;
  }

}
```

```
      }      return base * height / 2.0;
}
```

```
public class ShapeDemo
{
public static void main(String[] args){

Triangle t = new Triangle(6, 8);
System.out.println("Area is: " + t.calculateArea());

}
}
```

Interfaces

- Interface - specifies behavior for other classes
- It cannot be instantiated
- Methods listed in an interface must be written elsewhere
- Interface is like contract; it says "Any class that implements me needs
  to have a method for this and that…“
  implements keyword
- A class can implement any number of interfaces

```
 public class MyClass implements Comparable{
…
}
public class MyClass implements Comparable, ActionListener {
…
}
public class MyClass extends MySuperClass implements Comparable, ActionListener {
…
```

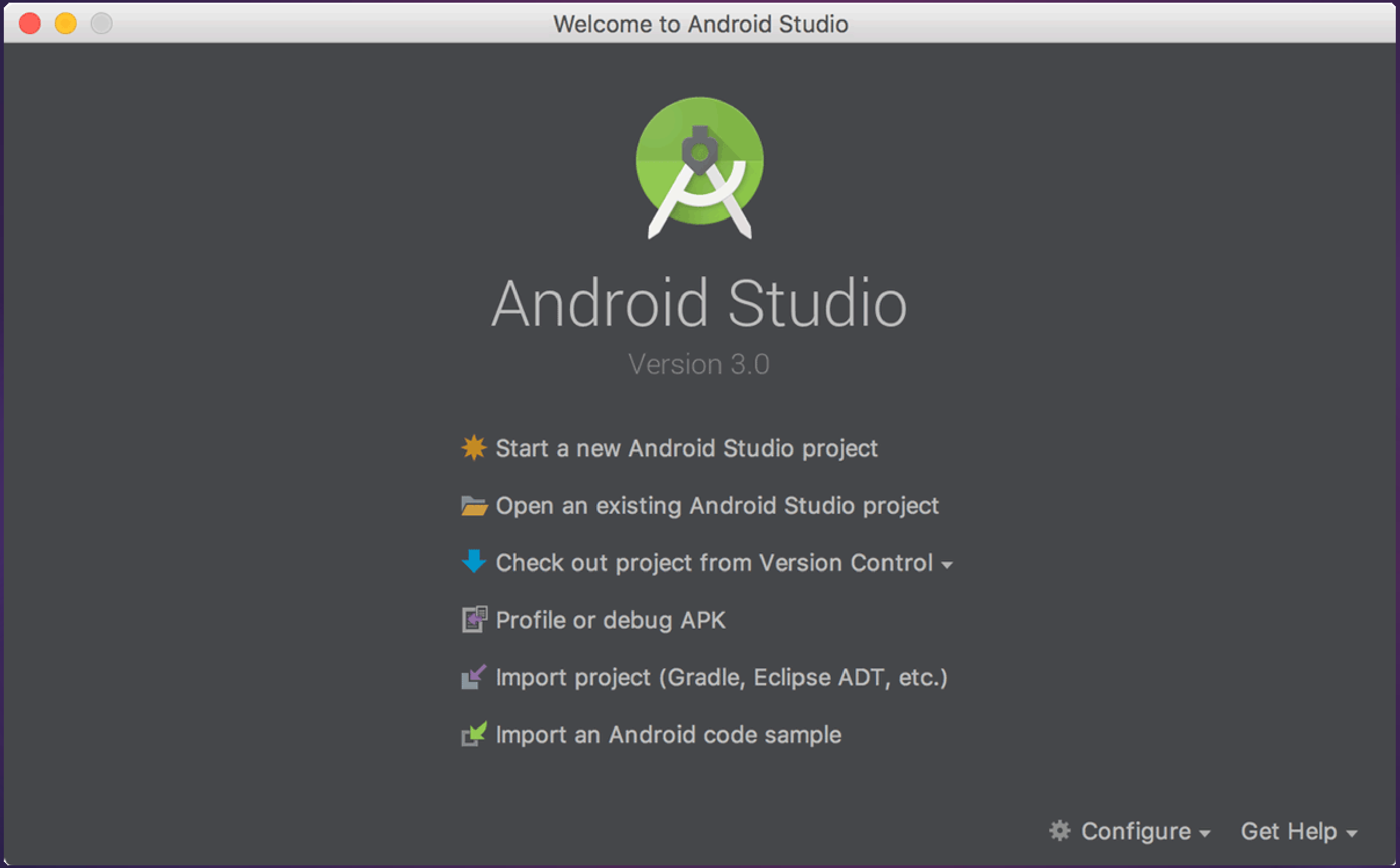
```
public class Stock implements Comparable
{

 private String name; private
 double sharePrice;

 public Stock(String n, double s)

 {
   name = n;

   sharePrice = s;

 }

 public String getName()



 {

   return name;

 }

 public double getSharePrice()

 {
   return sharePrice;

}
```

```
public void setSharePrice(double s)

{

sharePrice = s;

}

public int compareTo(Object obj)

{

Stock b = (Stock) obj; int result = 0;

//if the share prices are equal

if (b.getSharePrice() == getSharePrice())

{

result = 0;

}

//if the sharePrice is greater than b's sharePrice if (getSharePrice() >
b.getSharePrice())

{

result = 1;

}

//if the sharePrice is less than b's sharePrice

if (getSharePrice() < b.getSharePrice())

{

result = -1;
```

```
public class StockDriver

{


public static void main(String[] args)

{

Stock myStock = new Stock("XYZ", 200.0); Stock
myStock2 = new Stock("ABC", 200.0);


int val = myStock.compareTo(myStock2);


if (val == 0)

{

System.out.println(myStock.getName() + " is equals in
value as " + myStock2.getName());

}

if (val == -1)

{

System.out.println(myStock.getName() + " is lower in
value than " + myStock2.getName());


}

if (val == 1)

{

System.out.println(myStock.getName() + " is greater in
value than " + myStock2.getName());


}

}

}
```

Create an Android Project

1. In the **Welcome to Android Studio** window, click **Start a new Android Studio project**. Or if you have a project opened, select **File > New Project**.


Welcome to Android Studio

Android Studio

Version 3.0

Start a new Android Studio project

Open an existing Android Studio project

Check out project from Version Control

Profile or debug APK

Import project (Gradle, Eclipse ADT, etc.)

Import an Android code sample

Configure

Get Help

2. In the Create New Project window, enter the following values:

- Application Name: "My First App"
- Company Domain: "example.com“

You might want to change the project location. Also, if you want to write a Kotlin app, check the Include Kotlin support checkbox. Leave the other options as they are.

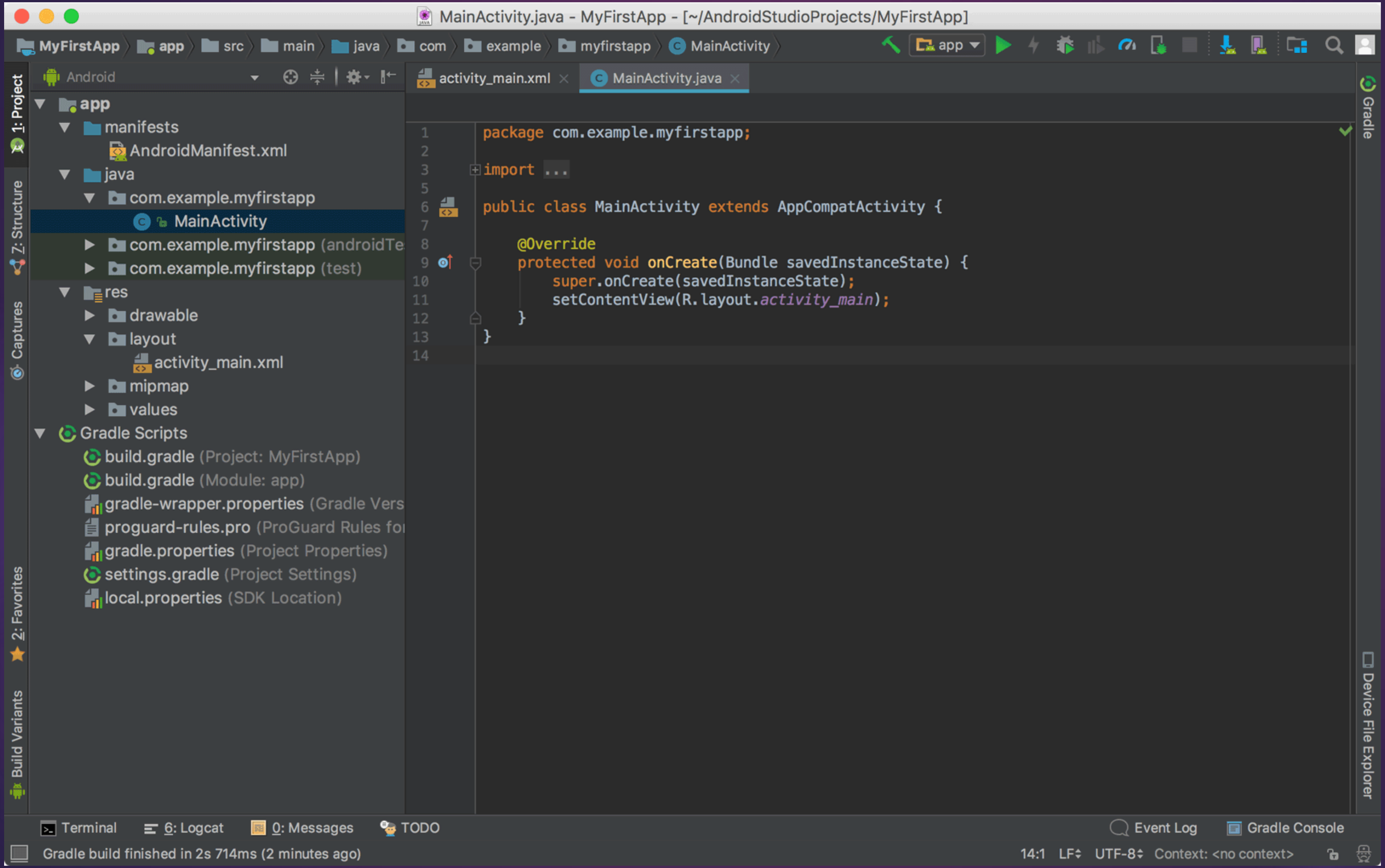
3. Click Next.

4. In the Target Android Devices screen, keep the default values and click Next.

5. In the Add an Activity to Mobile screen, select Empty Activity and click Next.

6. In the **Configure Activity** screen, keep the default values and click **Finish**.

After some processing, Android Studio opens the IDE.



MainActivity.java - MyFirstApp - [~/AndroidStudioProjects/MyFirstApp]

MyFirstApp › app › src › main › java › com › example › myfirstapp › MainActivity

- app
  - manifests
    - AndroidManifest.xml
  - java
    - com.example.myfirstapp
      - MainActivity
    - com.example.myfirstapp (androidTe
    - com.example.myfirstapp (test)
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
    - values
- Gradle Scripts
  - build.gradle (Project: MyFirstApp)
  - build.gradle (Module: app)
  - gradle-wrapper.properties (Gradle Vers
  - proguard-rules.pro (ProGuard Rules fo
  - gradle.properties (Project Properties)
  - settings.gradle (Project Settings)
  - local.properties (SDK Location)

activity_main.xml | MainActivity.java

```java
package com.example.myfirstapp;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Gradle build finished in 2s 714ms (2 minutes ago)

14:1 LF÷ UTF-8÷ Context: <no context>

First, be sure the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity
This is the main activity (the entry point for your app). When you build and run the app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml
This XML file defines the layout for the activity's UI. It contains a TextView element with the text "Hello world!".

app > manifests > AndroidManifest.xml
The manifest file describes the fundamental characteristics of the app and defines each of its components.

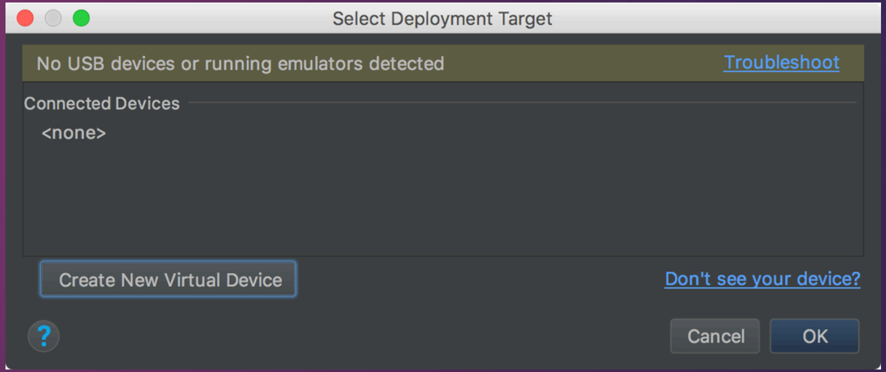
Gradle Scripts > build.gradle
You'll see two files with this name: one for the project and one for the "app" module. Each module has its own build.gradle file, but this project currently has just one module. You'll mostly work with the module's build.gradle file to configure how the Gradle tools compile and build your app.

Run Your App

Run on an emulator

Run the app on an emulator as follows:

1. In Android Studio, click the app module in the Project window and then select Run > Run (or click Run in the toolbar).
2. In the Select Deployment Target window, click Create New Virtual Device.
3. In the Select Hardware screen, select a phone device, such as Pixel, and then click Next.
4. In the System Image screen, select the version with the highest API level. If you don't have that version installed, a Download link is shown, so click that and complete the download.
5. Click Next.
6. On the Android Virtual Device (AVD) screen, leave all the settings alone and click Finish.
7. Back in the Select Deployment Target dialog, select the device you just created and click OK.


Select Deployment Target

No USB devices or running emulators detected

Connected Devices

<none>

Android Studio installs the app on the emulator and starts it. You should now see "Hello World!" displayed in the app running on the emulator.

Run on a real device

Set up your device as follows:

1. Connect your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device.

2. Enable USB debugging in the Developer options as follows.

First, you must enable the developer options:

a. Open the Settings app.

b. (Only on Android 8.0 or higher) Select System.

c. Scroll to the bottom and select About phone.

d. Scroll to the bottom and tap Build number 7 times.

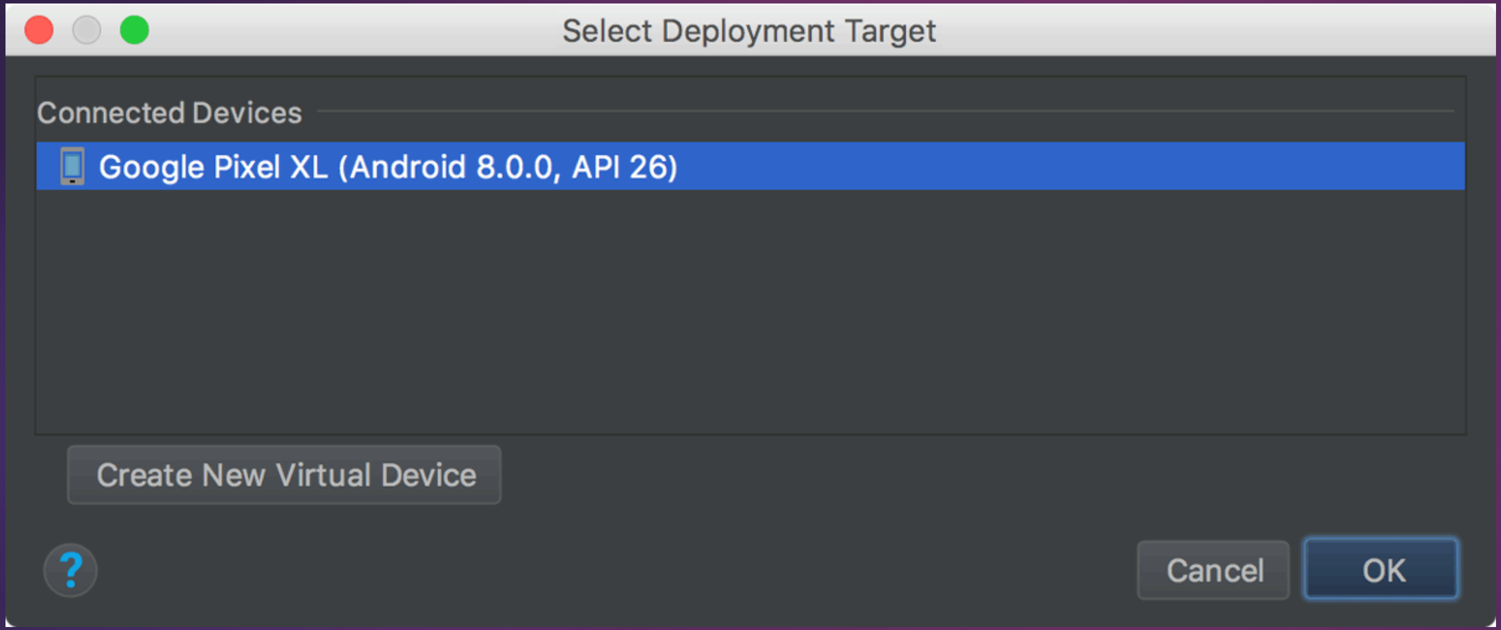
e. Return to the previous screen to find Developer options near the bottom.

Open Developer options, and then scroll down to find and enable USB debugging.

Run the app on your device as follows:

1. In Android Studio, click the app module in the Project window and then select Run > Run (or click Run in the toolbar).

2. In the Select Deployment Target window, select your device, and click OK.


Select Deployment Target
Connected Devices
Google Pixel XL (Android 8.0.0, API 26)
Create New Virtual Device
?
Cancel
OK

Android Studio installs the app on your connected device and starts it. You should now see "Hello World!" displayed in the app running on your device

Build a Simple User Interface

Build a Simple User Interface

In this lesson, you'll use the Android Studio Layout Editor to create a layout that includes a text box and a button. In the next lesson, you'll make the app respond to the button tap by sending the content of the text box to another activity.

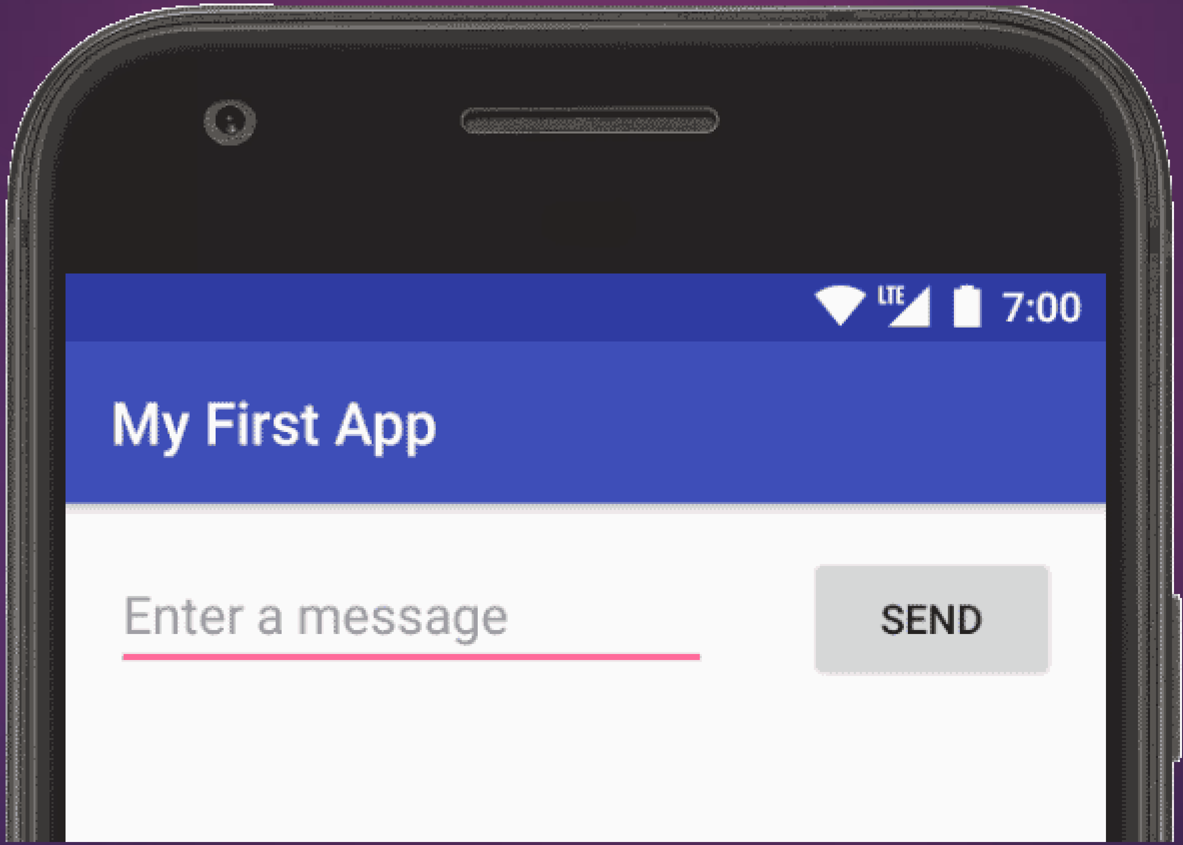


Figure 1. Screenshot of the final layout

The user interface for an Android app is built using a hierarchy of layouts (ViewGroup objects) and widgets (View objects). Layouts are invisible containers that control how its child views are positioned on the screen. Widgets are UI components such as buttons and text boxes.
Android provides an XML vocabulary for ViewGroup and View classes, so most of your UI is defined in XML files. However, instead of teaching you to write some XML, this lesson shows you how to create a layout using Android Studio's Layout Editor, which makes it easy to build a layout by drag-and-dropping views.

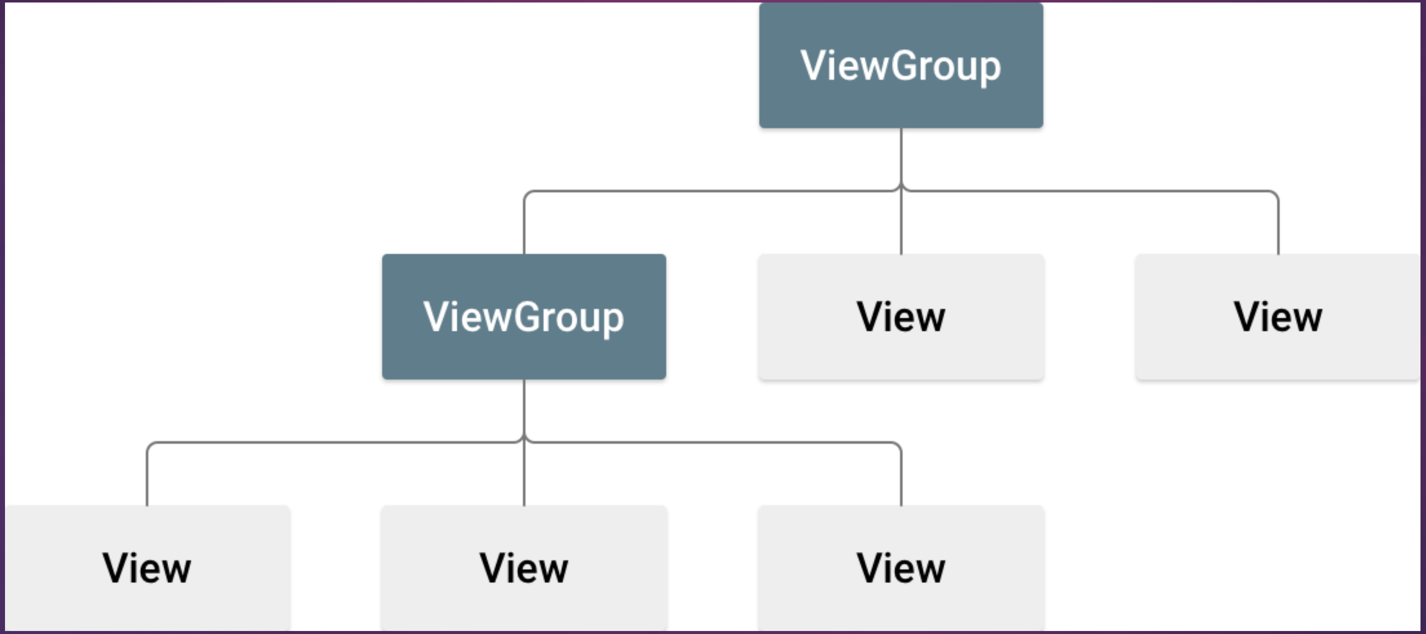
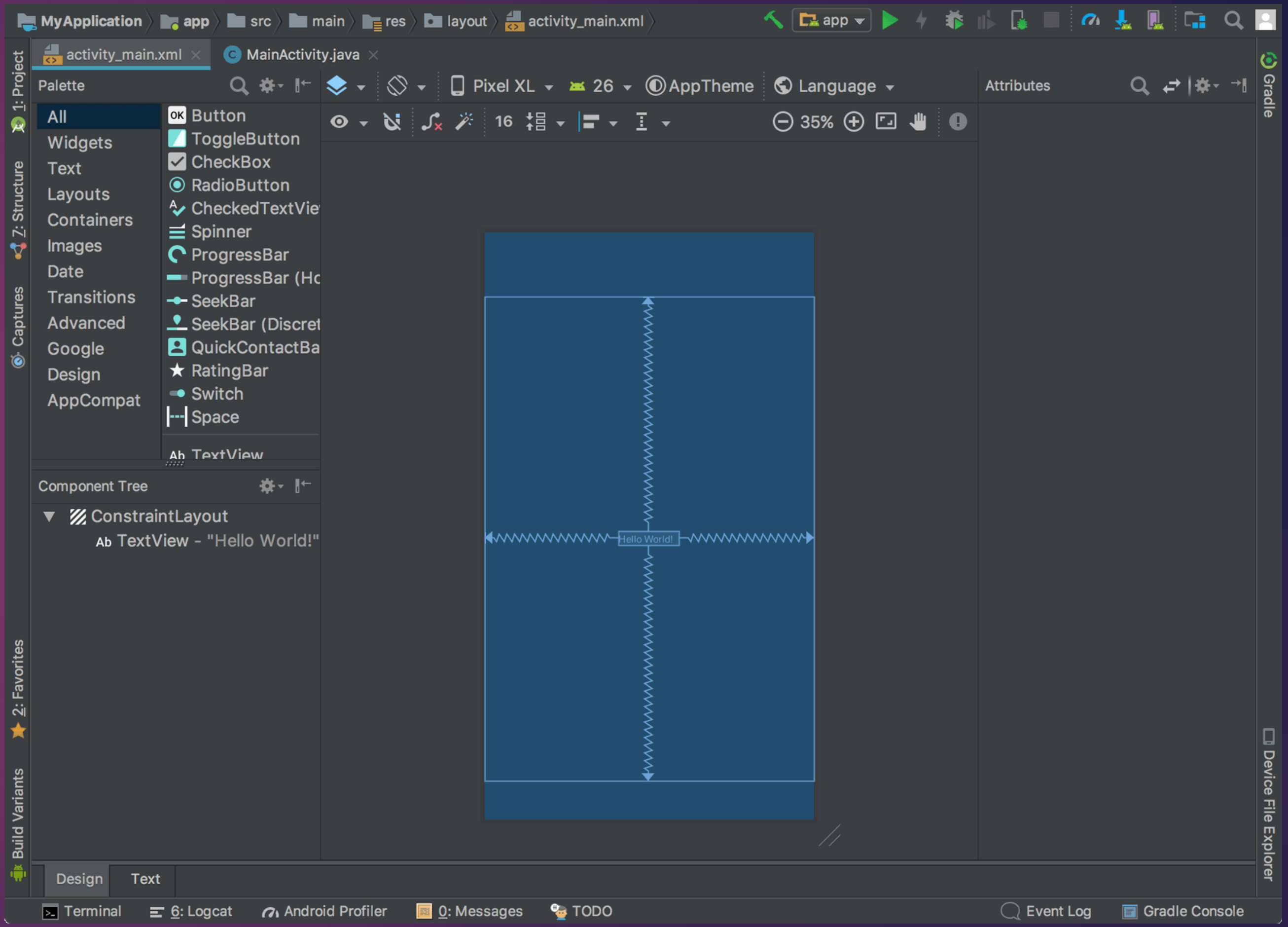


Figure 2: Illustration of how ViewGroup objects form branches in the layout and contain View objects

Open the Layout Editor

To get started, set up your workspace as follows:

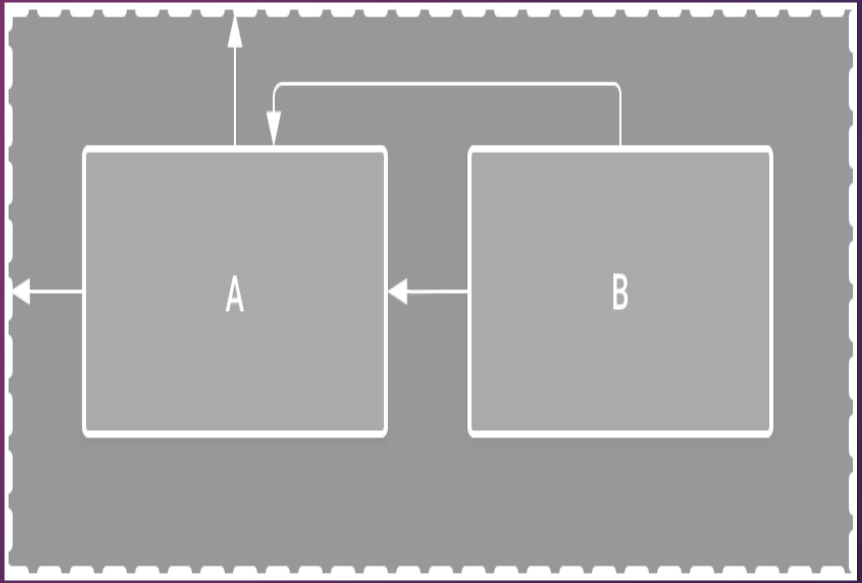
1. In Android Studio's Project window, open app > res > layout > activity_main.xml.
2. To make more room for the Layout Editor, hide the Project window by selecting View > Tool Windows > Project (or click Project on the left side of Android Studio).
3. If your editor shows the XML source, click the Design tab at the bottom of the window.
4. Click Select Design Surface and select Blueprint.
5. Click Show   in the Layout Editor toolbar and make sure Show Constraints is checked.
6. Make sure Autoconnect is off. The tooltip in the toolbar should read Turn On Autoconnect (because it's now off).
7. Click Default Margins in the toolbar and select 16 (you can still adjust the margin for each view later).
8. Click Device in Editor in the toolbar and select 5.5, 1440 × 2560, 560dpi (Pixel XL).



Open the Layout Editor

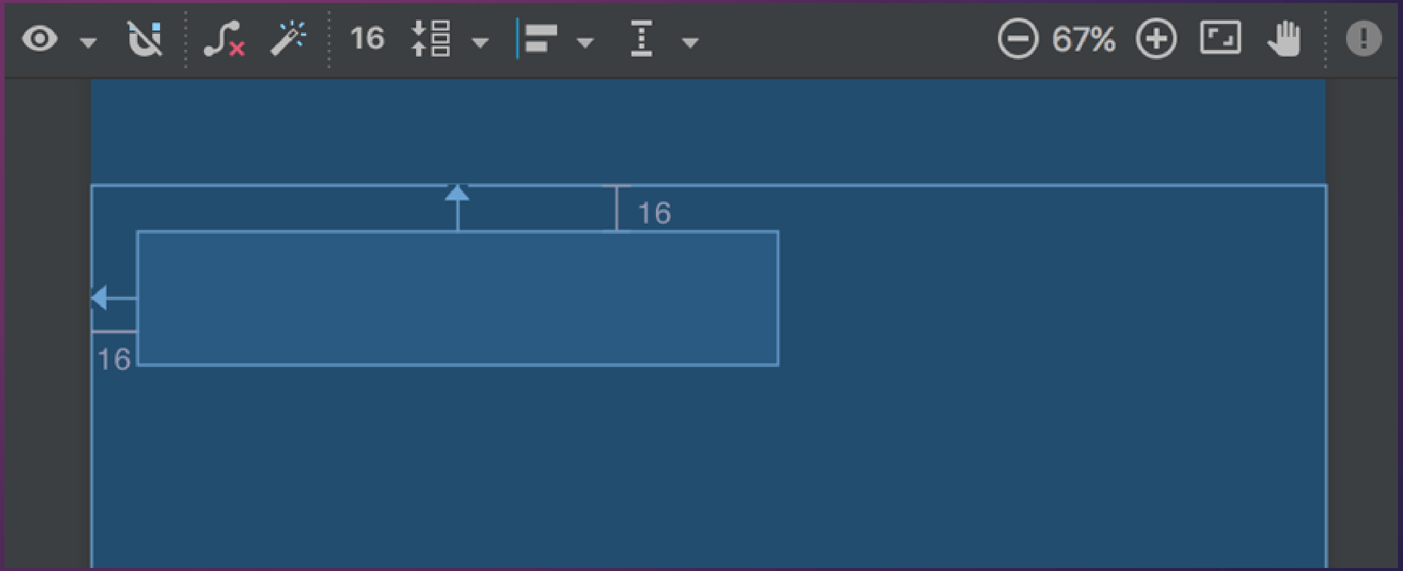
The Component Tree window on the bottom-left side shows the layout's hierarchy of views. In this case, the root view is a ConstraintLayout, containing just one TextView object.
ConstraintLayout is a layout that defines the position for each view based on constraints to sibling views and the parent layout. In this way, you can create both simple and complex layouts with a flat view hierarchy. That is, it avoids the need for nested layouts

- View A appears 16dp from the top of the parent layout.
- View A appears 16dp from the left of the parent layout.
- View B appears 16dp to the right of view A.
- View B is aligned to the top of view A



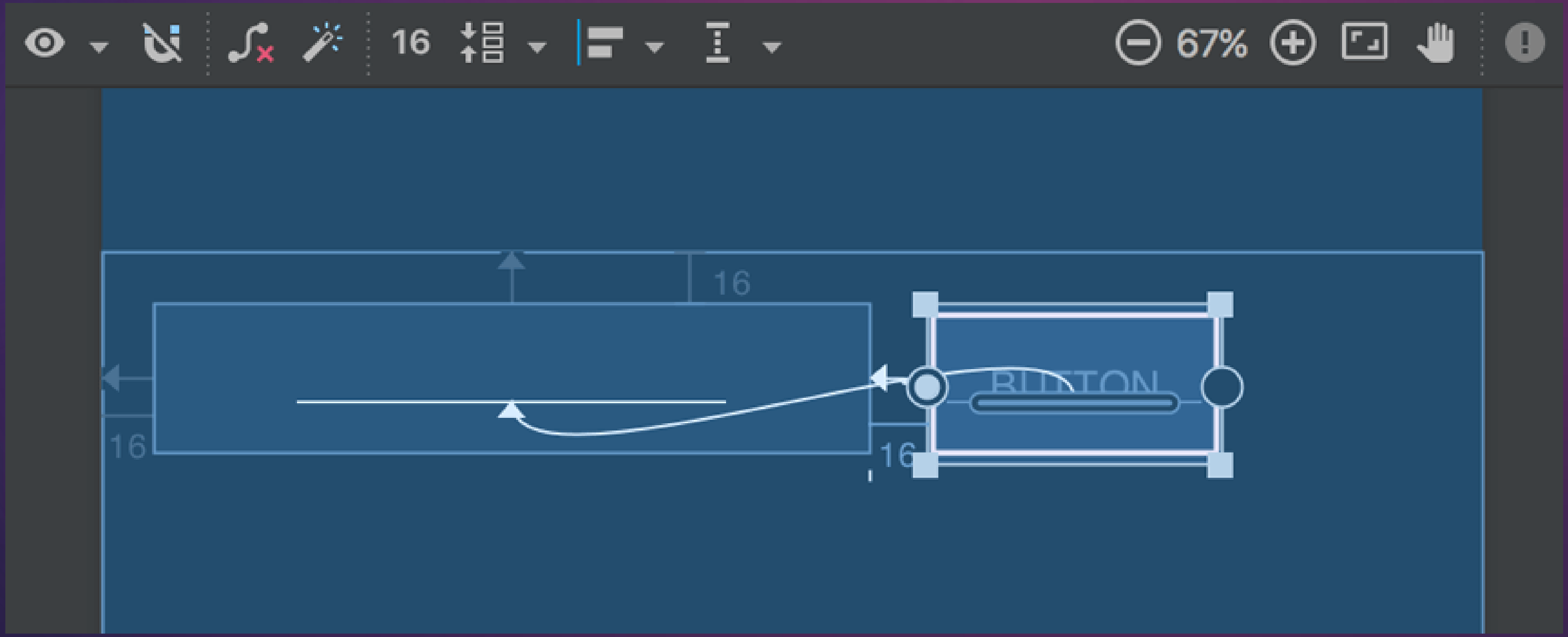
Add a text box

1. First, you need to remove what's already in the layout. So click **TextView** in the Component Tree window, and then press Delete.
2. In the Palette, click Text to show the available text controls.
3. Drag Plain Text into the design editor and drop it near the top of the layout. This is an **EditText** widget that accepts plain text input.
4. Click the view in the design editor. You can now see the resizing handles on each corner (squares), and the constraint anchors on each side (circles).
For better control, you might want to zoom in on the editor using the buttons in the Layout Editor toolbar.
5. Click-and-hold the anchor on the top side, and then drag it up until it snaps to the top of the layout and release. That's a constraint—it specifies the view should be 16dp from the top of the layout (because you set the default margins to 16dp).
6. Similarly, create a constraint from the left side of the view to the left side of the layout



Add a button

1. In the Palette, click Widgets.
2. Drag Button into the design editor and drop it near the right side.
3. Create a constraint from the left side of the button to the right side of the text box.
4. To constrain the views in a horizontal alignment, you need to create a constraint between the text baselines. So click the button, and then click Edit Baseline , which appears in the design editor directly below the selected view. The baseline anchor appears inside the button. Click-and-hold on this anchor and then drag it to the baseline anchor that appears in the text box.

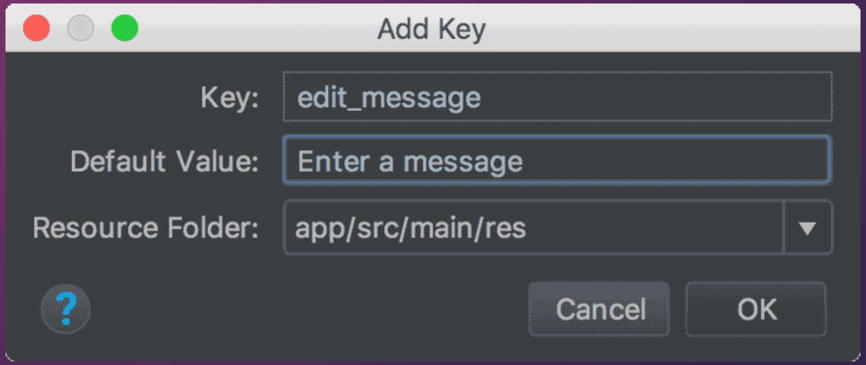


Note: You can also create a horizontal alignment using the top or bottom edges, but the button includes padding around its image, so the visual alignment is wrong if you align these views that way.

Change the UI strings

To preview the UI, click Select Design Surface in the toolbar and select Design. Notice that the text input is pre-filled with "Name" and the button is labeled "Button." So now you'll change these strings.
1. Open the Project window and then open app > res > values > strings.xml.
This is a string resources file where you should specify all your UI strings. Doing so allows you to manage all UI strings in a single location, which makes it easier to find, update, and localize (compared to hard-coding strings in your layout or app code).
2. Click Open editor at the top of the editor window. This opens the Translations Editor, which provides a simple interface for adding and editing your default strings, and helps keep all your translated strings organized.

3. Click Add Key to create a new string as the "hint text" for the text box.
a. Enter "edit_message" for the key name.
b. Enter "Enter a message" for the value.
c. Click OK.



Key: edit_message
Default Value: Enter a message
Resource Folder: app/src/main/res

Change the UI strings

4. Add another key named "button_send" with a value of "Send“

Now you can set these strings for each view. So return to the layout file by clicking activity_main.xml in the tab bar, and add the strings as follows:

a. Click the text box in the layout and, if the Attributes window isn't already visible on the right, click Attributes on the right sidebar.

b. Locate the text property (currently set to "Name") and delete the value.

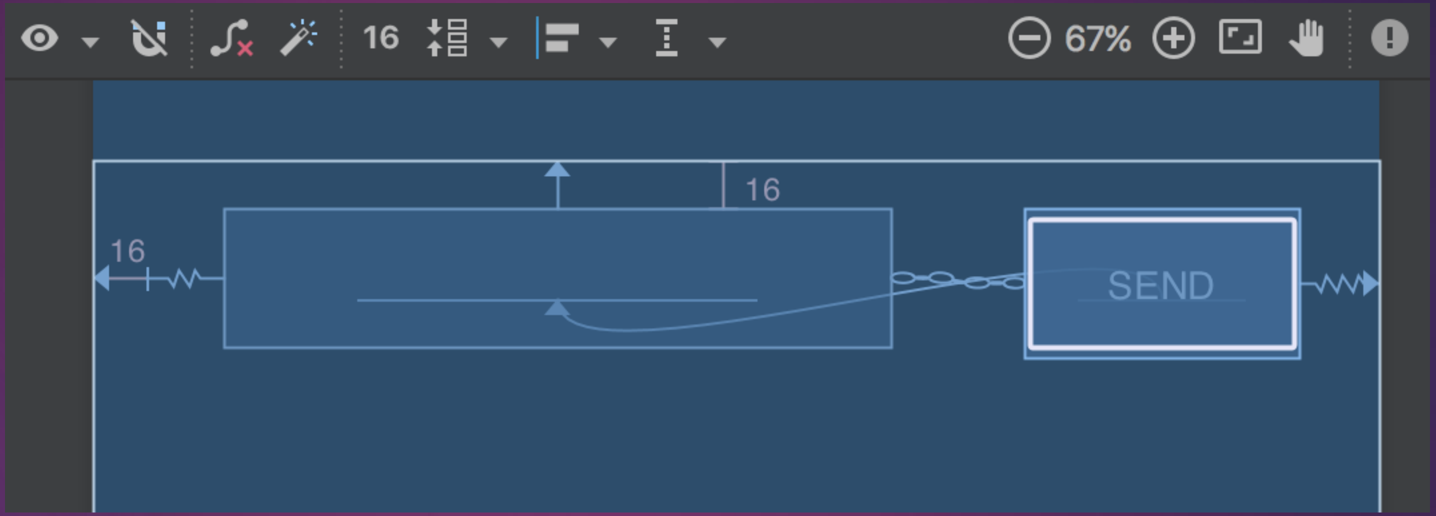
c. Locate the hint property and then click Pick a Resource to the right of the text box. In the dialog that appears, double-click on edit_message from the list.

d. Now click the button in the layout, locate the text property (currently set to "Button"), click Pick a Resource , and then select button_send.

Make the text box size flexible

To create a layout that's responsive to different screen sizes, you'll now make the text box stretch to fill all remaining horizontal space (after accounting for the button and margins).

1. Select both views (click one, hold Shift, and click the other), and then right-click either view and select Chain > Create Horizontal Chain. The layout should appear as shown.

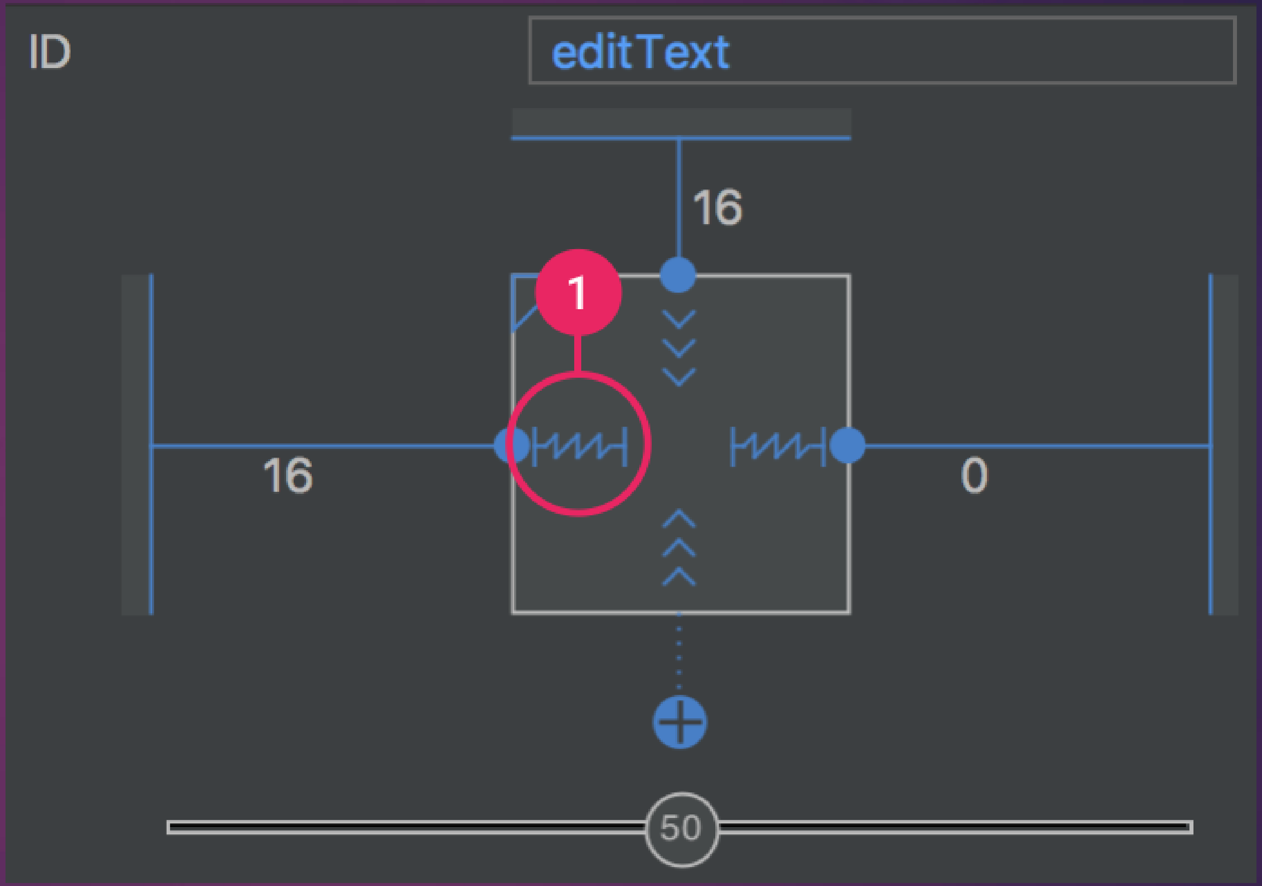


A chain is a bidirectional constraint between two or more views that allows you to lay out the chained views in unison.

2. Select the button and open the Attributes window. Using the view inspector at the top of the Attributes window, set the right margin to 16.

Make the text box size flexible

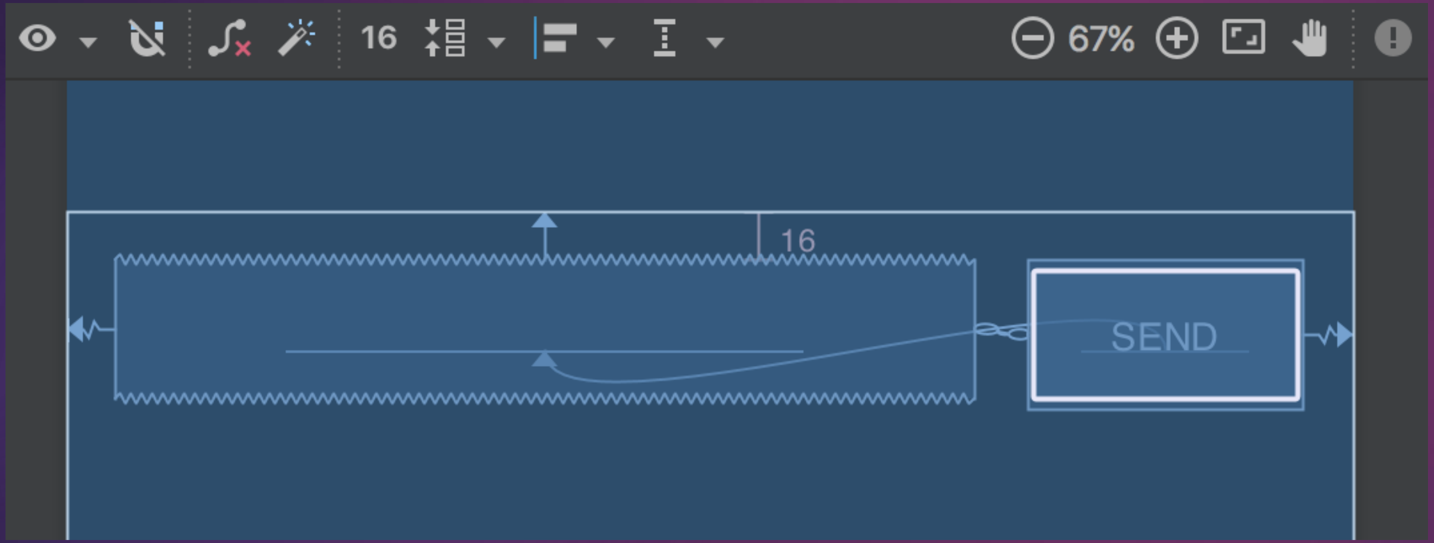
3. Now click the text box to view its attributes. Click the width indicator twice so that it is set to Match Constraints, as indicated by callout 1 in the figure.
"Match constraints" means that the width expands to meet the definition of the horizontal constraints and margins. Therefore, the text box stretches to fill the horizontal space (after accounting for the button and all margins).



Now the layout is done and should appear as shown.

Make the text box size flexible

Now the layout is done and should appear as shown.



Make the text box size flexible

If it seems your layout did not turn out as expected, click below to see what your the XML should look like and compare it to what you see in the Text tab. (If your attributes appear in a different order, that's okay.)

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.myfirstapp.MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:ems="10"
        android:hint="@string/edit_message"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toStartOf="@+id/button"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:layout_marginStart="16dp"
        android:text="@string/button_send"
        app:layout_constraintBaseline_toBaselineOf="@+id/editText"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/editText" />
</android.support.constraint.ConstraintLayout>
```

Run the app

If your app is already installed on the device from the previous lesson, simply click **Apply Changes** ⚡ in the toolbar to update the app with the new layout. Or click **Run** ▶ to install and run the app.

The button still does nothing. To start another activity when the button is tapped

Connection

Respond to the send button

Add a method to the MainActivity class that's called by the button as follows:

1. In the file app > java > com.example.myfirstapp > MainActivity, add the sendMessage() method stub as shown

You may see an error because Android Studio cannot resolve the View class used as the method argument. So click to place your cursor on the View declaration, and then perform a Quick Fix by pressing Alt + Enter (or Option + Enter on Mac). (If a menu appears, select Import class.)

```java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
    }
}
```

Respond to the send button

2. Now return to the activity_main.xml file to call this method from the button:

a. Click to select the button in the Layout Editor.
b. In the Attributes window, locate the onClick property and select sendMessage [MainActivity] from the drop-down list.

Now when the button is tapped, the system calls the sendMessage() method. Take note of the details in this method that are required in order for the system to recognize it as compatible with the android:onClick attribute. Specifically, the method has the following characteristics:

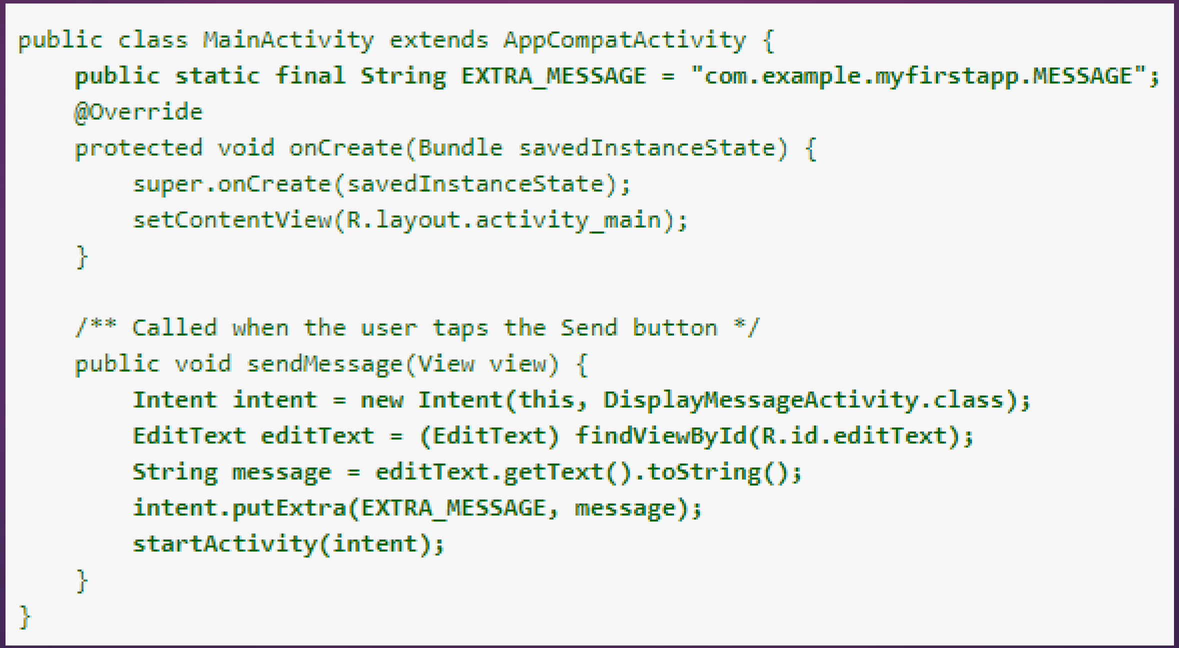
- Public access
- A void or, an implicit unit return value
- A View as the only parameter (it is the View object that was clicked)

Next, you'll fill in this method to read the contents of the text field and deliver that text to another activity

Build an Intent

An Intent is an object that provides runtime binding between separate components, such as two activities. The Intent represents an app’s "intent to do something." You can use intents for a wide variety of tasks, but in this lesson, your intent starts another activity.

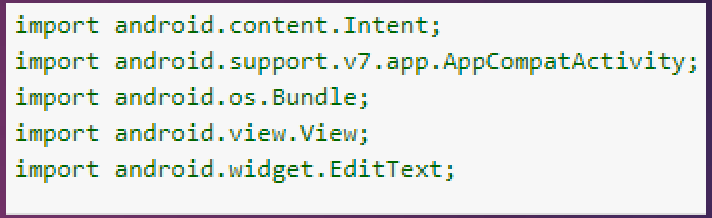
In MainActivity, add the EXTRA_MESSAGE constant and the sendMessage() code, as shown here



```java
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.editText);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

Android Studio again encounters Cannot resolve symbol errors, so press Alt + Enter (or Option + Return on Mac). Your imports should end up as the following

An error remains for DisplayMessageActivity, but that's okay; you'll fix that in the next section.



```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
```

Here's what's going on in sendMessage():

The Intent constructor takes two parameters:
A Context as its first parameter (this is used because the Activity class is a subclass of Context)
The Class of the app component to which the system should deliver the Intent (in this case, the activity that should be started).
The putExtra() method adds the EditText's value to the intent. An Intent can carry data types as key-value pairs called extras. Your key is a public constant EXTRA_MESSAGE because the next activity uses the key to retrieve the text value. It's a good practice to define keys for intent extras using your app's package name as a prefix. This ensures the keys are unique, in case your app interacts with other apps.
The startActivity() method starts an instance of the DisplayMessageActivity specified by the Intent. Now you need to create that class

Create the second activity

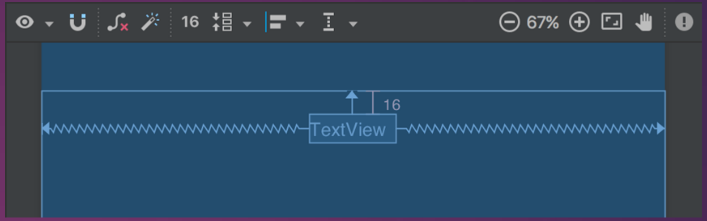
1. In the Project window, right-click the app folder and select New > Activity > Empty Activity.
2. In the Configure Activity window, enter "DisplayMessageActivity" for Activity Name and click Finish (leave all other properties set to the defaults).
Android Studio automatically does three things:

- Creates the DisplayMessageActivity file.
- Creates the corresponding activity_display_message.xml layout file.
- Adds the required <activity> element in AndroidManifest.xml.

If you run the app and tap the button on the first activity, the second activity starts but is empty. This is because the second activity uses the empty layout provided by the template

Add a text view

The new activity includes a blank layout file, so now you'll add a text view where the message will appear.



1. Open the file app > res > layout > activity_display_message.xml.
2. Click Turn On Autoconnect in the toolbar (it should then be enabled, as shown in figure 1).
3. In the Palette window, click Text and then drag a TextView into the layout—drop it near the the top of the layout, near the center so it snaps to the vertical line that appears. Autoconnect adds left and right constraints to place the view in the horizontal center.
4. Create one more constraint from the top of the text view to the top of the layout, so it appears as shown in figure

Optionally, make some adjustments to the text style by expanding textAppearance in the Attributes window and change attributes such as textSize and textColor.

Display the message

Now you will modify the second activity to display the message that was passed by the first activity.

1. In DisplayMessageActivity, add the following code to the onCreate() method

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_display_message);

    // Get the Intent that started this activity and extract the string
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Capture the layout's TextView and set the string as its text
    TextView textView = findViewById(R.id.textView);
    textView.setText(message);
}
```

2. Press Alt + Enter (or Option + Return on Mac) to import missing classes. Your imports should end up as the following

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

Add up navigation

Each screen in your app that is not the main entry point (all screens that are not the "home" screen) should provide navigation so the user can return to the logical parent screen in the app hierarchy by tapping the Up button in the app bar.

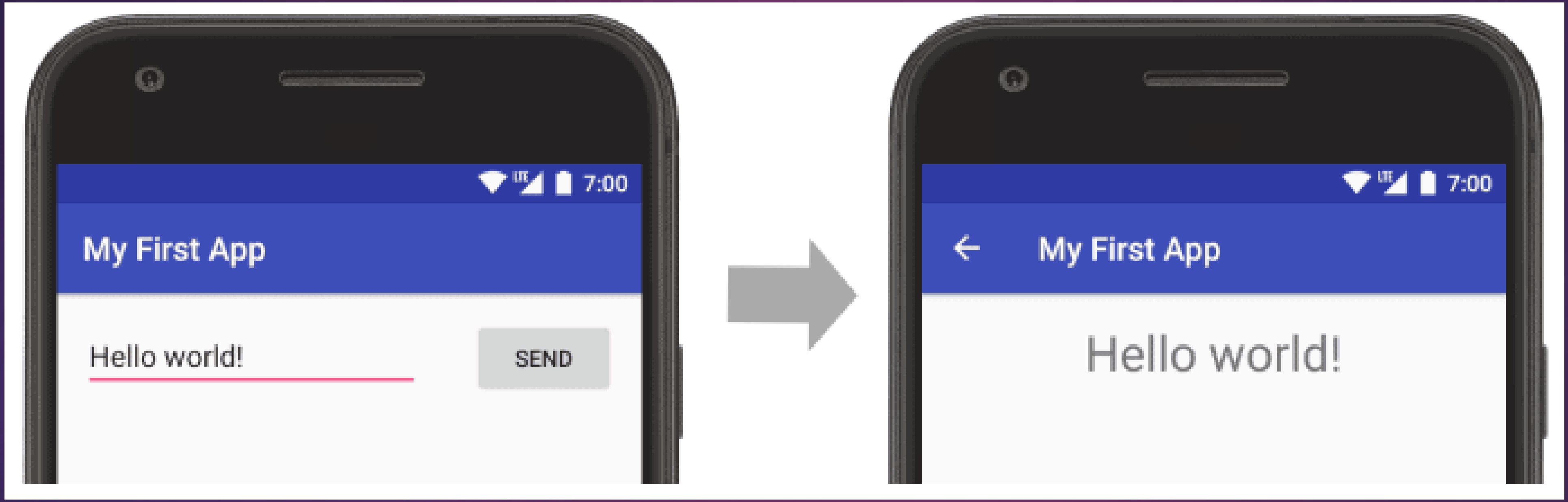
All you need to do is declare which activity is the logical parent in the AndroidManifest.xml file. So open the file at app > manifests > AndroidManifest.xml, locate the <activity> tag for DisplayMessageActivity and replace it with the following:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

The Android system now automatically adds the Up button in the app bar

Run the app

Now run the app again by clicking Apply Changes in the toolbar. When it opens, type a message in the text field, and tap Send to see the message appear in the second activity.


7:00
My First App
Hello world!
SEND
7:00
My First App
Hello world!