PROG 10082
Object Oriented Programming 1

Recap

Reading Input from the Console

1. Create a Scanner object

```
Scanner input = new Scanner(System.in);
```

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value. For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

ComputeAreaWithConsoleInput

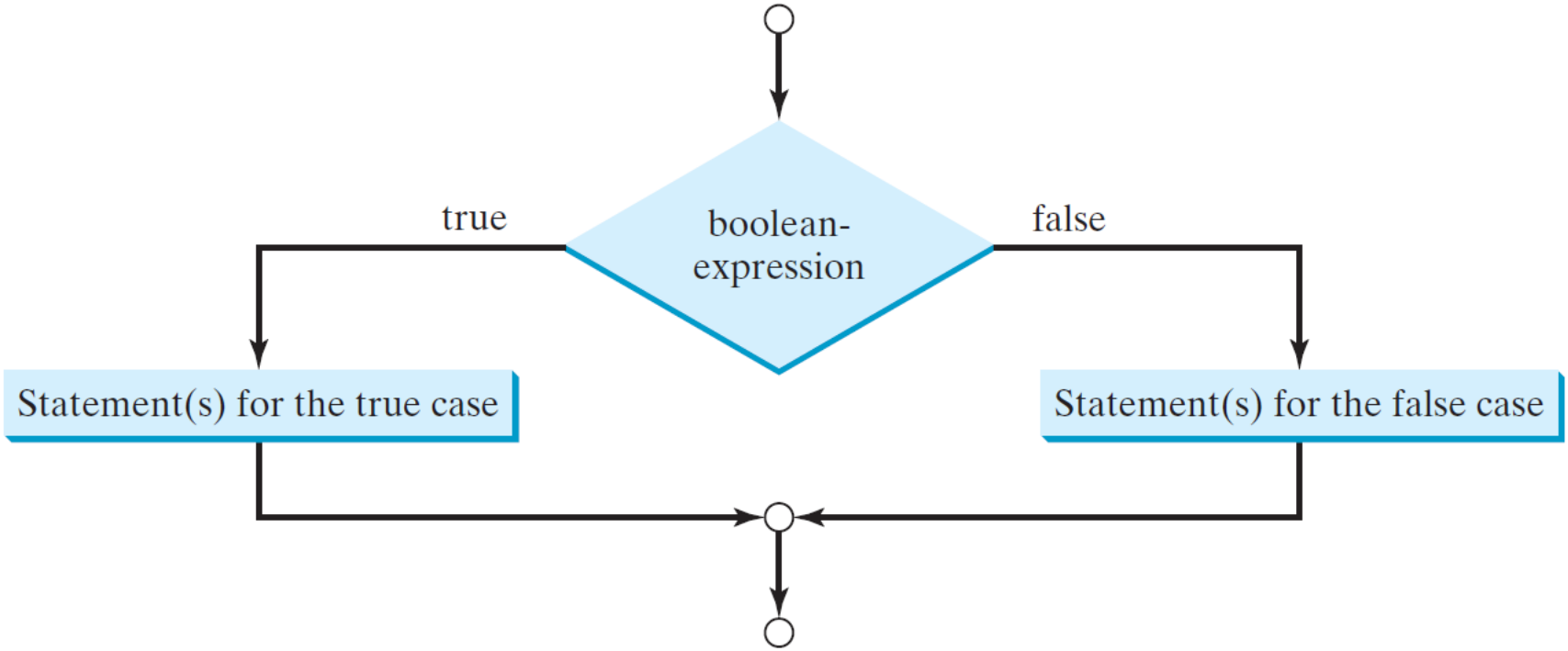
Run

ComputeAverage

Run

The Two-way `if` Statement

```
if (boolean-expression) {
  statement(s)-for-the-true-case;
}
else {
  statement(s)-for-the-false-case;
}
```



Single-Sided Ifs – Exercise

Write a program that asks the user for a number less than 100. If the user's input is 100 or more, display a message on the console.

```java
import java.util.Scanner;

public class ValidNumber1 {

    public static void main(String[] args) {

        // construct a scanner for user input
        Scanner in = new Scanner(System.in);

        // prompt user for number less than 100
        System.out.print("Enter a number less than 100:");
        int num = in.nextInt();

        // if user input number is invalid
        if (num >= 100) {
            // display error msg
            System.out.println("Error: invalid number.");
        }
    }
}
```

2. Modify the program: Instead of checking to see if the number is 100 or more, check to see if the number is less than 100. If it is, calculate the square root of the number, the square of the number, and the cube of the number. Then display these on the console.

```java
import java.util.Scanner;

public class ValidNumber2 {

    public static void main(String[] args) {

        // construct a scanner for user input
        Scanner in = new Scanner(System.in);

        // prompt user for number less than 100
        System.out.print("Enter a number less than 100:");
        int num = in.nextInt();

        // if the number is valid display stuff
        if (num < 100) {

            // calculate square, root, and cube
            double square = Math.pow(num, 2);
            double root = Math.sqrt(num);
            double cube = Math.pow(num, 3);

            // display results of calculations, formatted
            System.out.printf("%-7s: %.2f%n", "Square", square);
            System.out.printf("%-7s: %.2f%n", "Root", root);
            System.out.printf("%-7s: %.2f%n", "Cube", cube);
        }
    }
}
```

Double-Sided Ifs – Exercise

Write programs for each of the following:

1) Ask the user for a whole number. Display a message on the screen that indicates whether the number is even or odd.
2) The method Math.random() generates a random double number greater than 0.0 and less than 1.0. Write a program that simulates flipping a coin. Call the Math.random() method and if the number generated is less than .5, display "HEADS". Otherwise, display "TAILS".
3) Write a program that generates a random integer from 1 to 5. Then ask the user to guess that number. If they guess correctly, display the message, "You guessed correctly!". If they guess incorrectly, display the message, "Sorry, you guessed incorrectly."
4) Write a program that simulates the tossing of two dice (generate two random integers from 1 to 6). If doubles are rolled, display a "You Win!!" message. Otherwise, tell the user what dice values they rolled.
5) In a bank where you work, money is given an interest rate of 7.5% if it is left in an account for more than 5 years; otherwise, the interest rate is 4.5%. Write a program that asks the user for their current account balance, and number of years their money has been sitting in the account. Then assign the proper interest rate to a variable interestRate. Finally, calculate and display the total interest for the current year

6) A program calculates a person's weekly pay based on the following: If the hours worked are less than or equal to 40, the person receives 10 dollars per hour; otherwise, the person receives 10 dollars for the first 40 hours and then 15 dollars for the hours they work over 40. Your program should ask for the number of hours from the user, and then calculate the weekly pay. Display your result.

7) Write a program that checks for a valid department number entered by the user. A department number is valid if it is between 1 and 45, inclusive. Display messages stating whether or not the department number is valid.

8) Modify the previous question: A valid department number is between 1 and 45 inclusive, or greater than 200.

9)Modify the previous question: valid department numbers include department 50, those between 1 and 45 inclusive, and those that are greater than 200.

Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

Marginal Tax Rate	*Single*	*Married Filing Jointly or Qualifying Widow(er)*	*Married Filing Separately*	*Head of Household*
10%	$0 – $8,350	$0 – $16,700	$0 – $8,350	$0 – $11,950
15%	$8,351 – $33,950	$16,701 – $67,900	$8,351 – $33,950	$11,951 – $45,500
25%	$33,951 – $82,250	$67,901 – $137,050	$33,951 – $68,525	$45,501 – $117,450
28%	$82,251 – $171,550	$137,051 – $208,850	$68,526 – $104,425	$117,451 – $190,200
33%	$171,551 – $372,950	$208,851 – $372,950	$104,426 – $186,475	$190,201 – $372,950
35%	$372,951+	$372,951+	$186,476+	$372,951+

Problem: Computing Taxes, cont.

```
if (status == 0) {
  // Compute tax for single filers
}
else if (status == 1) {
  // Compute tax for married file jointly
  // or qualifying widow(er)
}
else if (status == 2) {
  // Compute tax for married file separately
}
else if (status == 3) {
  // Compute tax for head of household
}
else {
  // Display wrong status
}
```

ComputeTax Run

Exercise

1)Find the Largest Number Among Three Numbers **using if-else..if**. Prompt the user to enter the values

Prcosedure:

1. Take three integer variables, say A, B& C
2. Assign values to variables
3. If A is greater than B & C, Display A is the largest value
4. If B is greater than A & C, Display B is the largest value
5. If C is greater than A & B, Display A is the largest value
6. Otherwise, Display A, B & C are not unique values

2) Rewrite the above program **using nested if**.

Boolean Data Type

The boolean primitive data type is used for true/false values. You can write various expressions that result in a boolean, and you can also create boolean variables.

All relational and logical expressions must result in a boolean value of true of false.

Boolean primitive variables can be declared and used in a program, but they will only accept the values **`true`** and **`false`**. You can't assign any other value to a boolean variable.

Additionally, the Scanner class contains a nextBoolean() method that allows you to retrieve a true/false value from the input source:

```
1   import java.util.Scanner;
2
3  public class BoolStuff {
4
5    public static void main(String[] args) {
6
7        Scanner in = new Scanner(System.in);
8        System.out.println("True or False: Java was invented by a Canadian.");
9        boolean answer = in.nextBoolean();
10
11       System.out.println("Your answer: " + answer);
12   }
13 }
```

In the program above, the user can type the word "true" or the word "false", and they can type it in any case, and the nextBoolean() method will return the boolean value "true" or "false", as appropriate.

For example, try the program - run it and try typing "TRUE", "False", "tRUE", and "FaLsE". Your program will only output the boolean values true or false (in all lower-case).

If the value entered by the user can't be translated into a boolean value, then an InputMismatchException occurs and the program crashes (much like it would if you asked for an int or double and the user entered letters).

One way to avoid errors is to grab the boolean input as a String using the Scanner's next() method. Then you can use the parseBoolean() method from the Boolean class (this is a "wrapper class" for the boolean data type).

The Boolean.parseBoolean() method is a static method (invoke it using the name of the class Boolean) and it receives a String input/argument. It will examine that string input and turn it into a boolean primitive value: If the string looks like the word "true", then it will return the boolean value true. If the string looks like anything else ("false", "apple", "Fred", etc) then it will return the boolean value false.

```
1import java.util.Scanner;
2
3public class BoolStuff {
4
5    public static void main(String[] args) {
6
7        Scanner in = new Scanner(System.in);
8        System.out.println("True or False: Java was invented by a Canadian.");
9        boolean answer = in.next();
10
11       System.out.println("You typed " + answer);
12
13       boolean boolAnswer = Boolean.parseBoolean(answer);
14       System.out.println("That is " + boolAnswer);
15   }
16}
```

Test the above program: try typing valid boolean values, and try typing some nonsense values (e.g. "apple")

Relational Expression

A **relational expression**, sometimes also referred to as a **conditional expression** or **condition**, includes two operands and one of the relational operators.
The relational operators might be represented differently in different languages.

For example, != in Java is sometimes written as **NOT=**, **<>**, or **~=**. Also, in some languages, the == relational operator (pronounced "double equals") is written as =.
A simple example using the double-equals and not-equals operator in Java:

```
int someVar = 5; int otherVar = 7;
System.out.println(someVar != otherVar);
System.out.println(someVar == otherVar);
```

In the first and second statements, we are assigning the value 5 to someVar and the value 7 to otherVar. In the third statement, we are asking if someVar contains a value that is different from the value in otherVar.

In this case, the relational expression is **someVar != otherVar** and its result is the boolean value true. This value is sent to the println() and is displayed on the console. The fourth statement contains the relational expression **someVar == otherVar**, which results in the boolean value false. Thus, "false" is the output for this statement.

Alternatively, you could have placed the result of these two relational expressions into boolean variables:

```
int someVar = 5; int otherVar = 7;
boolean firstStmt = someVar != otherVar;
boolean secondStmt = someVar == otherVar;
System.out.println(firstStmt);
System.out.println(secondStmt);
```

It is important to note that **relational expressions always result in a boolean value of true or false**.

For example given that varX = 3 and varY = 9, what would be the value of each of the following relational expressions?

varX > 3 ?	false
varX > varY ?	false
varX <= varY ?	true
varY = 5 ?	Y = 5 is actually an assignment statement, and we know 5 is an int.. so if Y is also an int, then this has an int result.

Exercises - 1

1. Which of the following are relational expressions in Java? For the items that aren't relational expressions, what kind of expressions/statements are they?

a.x == 3
b.x = 3
c.x >= 3
d.x * 3
e.3 < x
f.x - 3 <= 10

2. Write a program that generates two random integers between 1 and 10 and store them into the variables value1 and value2.

There are a couple of ways to generate a random integer. One easy way is to use the Math.random():

```
int randomNum = (int)(Math.random() * 100) + 1;
```

The statement above will generate a random number between 1 and 100.

After generating two random numbers and storing them into variables, display a math quiz by asking the user what value1 + value2 is. Ask the user for their answer, and display whether their answer is "true" or "false". Sample user interaction is shown below (user input in **purple**):

```
What is 4 + 3? 7 4 + 3 = 7 is true
```

Another output example:

```
What is 2 + 1? 2 2 + 1 = 2 is false
```

Comparing Strings

Comparing strings in Java is not as straightforward as it might seem at first.

Strings are objects, so we shouldn't compare them with the == operator. With objects, comparing with == checks to see if two variables are "pointing to" the same object in memory.

To compare strings with ==. Instead, we use a special method called equals().

The equals() method is part of Java. It operates on a single string, and you pass a second string to it as an argument.

The equals() method will then compare the two strings for equality.

If the strings are the same, the method will return the boolean value true.

If the strings are not equal, the method will return the boolean value false

Example:

```
tring name = "Kaluha";
String userName = "Fred";
String beverage = "kaluha";
System.out.println(name.equals(userName)); // prints false
System.out.println(name.equals("Kaluha")); // prints true
System.out.println(name.equals(beverage)); // prints false
```

In the first print statement, the string "Kaluha" is being compared to "Fred". These are not equal, so the result false is printed.

In the second print statement, the string "Kaluha" is being compared with the string "Kaluha". These are the same, so the result true is printed.

In the last print statement, the string "Kaluha" is not equal to "kaluha", so the result false is printed.

This last print statement brings up an important issue: what if you want a case-insensitive comparison?

Use the equalsIgnoreCase() method instead! For example, the statement

```
System.out.println(name.equalsIgnoreCase(beverage));
```

will print the boolean result true!

Exercise - 2

1. Write a program that asks the user for a name. Display the result of comparing the user-entered name to your own name.

logical operators

Operator	Symbol
AND	&&
OR	||
NOT	!

The expressions used above could all be written as:

```
deptNum >= 1 && deptNum <= 10 deptNum < 1 || deptNum > 10
!(deptNum > 0)
```

Note that there must be a complete expression on each side of the logical operator. In other words, the statement "deptNum >= 1 && <= 10" is invalid!

Logical operators have an order of precedence! Order of precedence for logical operators are () (brackets), NOT, AND, OR. Be sure to refer to your Order of Precedence chart in Appendix C.

Exercise - 3