The information we manage in a Java program is either represented as *primitive data* or as *objects*.

- **Primitive data** (נתונים פרימיטיביים) include common, fundamental values as numbers and characters.

- Java is an object-oriented language and the **object** (Ӻצם) is a fundamental entity in Java programming. The operations that can be performed on the object are defined by the **methods** in the class.

- A **method** (פעולה/שיטה) is the object-oriented term for a procedure or a function. Treat it as a synonym for "procedure." The data and methods, taken together, usually serve to define the contents and capabilities of some kind of object.

- A **class** (מחלקה) is a collection of data and methods that operate on that data. An object is defined by a class.
Our first Java program

```java
import java.util.*;

/* HelloWorld – An example Java program */

public class MyFirst // class name
{
    public static void main(String[ ] args)
    {
        System.out.println("Hello, world!");
    }
}
```

- **import** java.util.*: allows Java libraries (classes) to be referenced.
- /* HelloWorld – An example Java program */: desirable comments.
- **public class** MyFirst // class name: class name.
- **public static void** main(String[ ] args): main(...) method means “start here”.
- System.out.println("Hello, world!"): all Java statements end with a semicolon (;).
- both, class and method, are delimited by braces.
### The primitive types in Java

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>(-2^7) (-128) to (2^7)-1 (127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>(-2^{15}) (-32768) to (2^{15})-1 (32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>(-2^{31}) (-2147483648) to (2^{31})-1 (2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>(-2^{63}) to (2^{63})-1 (i.e., -9223372036854775808 to 9223372036854775807)</td>
<td>64-bit signed</td>
</tr>
</tbody>
</table>
| float | **Negative range:** 
-3.4028235E38 to -1.4E-45  
**Positive range:** 
1.4E-45 to 3.4028235E38 | 32-bit IEEE 754 |
| double| **Negative range:** 
-1.7976931348623157E308 to -4.9E-324  
**Positive range:** | 64-bit IEEE 754 |
| char  | single character                  | 16-bit       |
Java characters use **Unicode**, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages.

Unicode takes **two bytes**, preceded by `\u`, expressed in four hexadecimal numbers that run from `'\u0000'` to `'\uFFFF'`. So, Unicode can represent 65535 + 1 characters.

**Example:**

Unicode `\u03b1 \u03b2 \u03b3` for three Greek letters
ASCII Character Set

ASCII Character Set is a **subset of the Unicode**
from \u0000 to \u007f

For example:  A = 65H = 0110 0101b;  B = 66H = 0110 0110b
A **variable** is a name for a location in memory used to hold a data value.

```java
// Compute the first area
double radius = 1.0;
double area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);
```
Declaring variables

- A variable declaration instructs the compiler to reserve a portion of main memory space large enough to hold a particular type of value and indicates the name by which we refer to that location.

```c
int x;  // Declare x to be an integer variable
double radius;  // Declare radius to be a double variable
char a1;  // Declare a to be a character variable
```

- Each variable can be initialized in the declaration.
- Variable declaration can have multiple variables of the same type declared on one line.

```c
double num1, num2=4.12, num3=2.89;
char ch1='a', ch2;
```
A boolean value, defined in Java using the reserved word `boolean`, has only two valid values: `true` and `false`.

A boolean value cannot be converted to any other data type, nor can any other data type be converted to a boolean value.

The words `true` and `false` are reserved in Java as boolean literals and cannot be used outside of this context.

**Examples** of boolean variable declarations in Java:

```java
boolean flag = true;
boolean a1, a2 = false, a3;
```
Identifiers naming roles

- An identifier is a sequence of characters that consist of letters, digits, underscores (\_), and dollar signs ($).
- An identifier **must start with a letter**, an underscore (\_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a **reserved word** (מילים שמורות).
- **Case sensitive**: A1 is different from a1
- An identifier can be of any length.
1. Create a Scanner object:

   `static Scanner reader = new Scanner(System.in);`

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

   **For example:**

   
   `static Scanner reader = new Scanner(System.in);`

   ```
   System.out.print("Please enter a double value: ");
   double num1 = reader.nextDouble();
   ```

   ```
   System.out.print("Please enter an integer value: ");
   int num2 = reader.nextInt();
   ```
import java.util.*;
class Example1
{
    static Scanner reader = new Scanner(System.in);
    public static void main(String[ ] args)
    {
        int num1,num2;
        double num3;
        System.out.println("Enter 2 integers :");
        num1 = reader.nextInt();
        num2 = reader.nextInt();
        System.out.print ("Enter double number: ");
        num3 = reader.nextDouble();
        System.out.println("num1+num2= "+(num1+num2));
        System.out.println("num3= "+num3);
    }
}
Java statements 1

- A Java method body is a series of zero or more *statements*. **Statement** (הוראה) is an instruction to the computer to do something.
- In the Java programming language statements are the fundamental unit of execution. All statements except blocks are terminated by a semicolon. Blocks are denoted by open and close curly braces.
- Statements are executed for their effects; they do not have values.
- Statements generally contain *expressions* (ביטויים).
- expression contains *operators* and *operands* (אופרטורים ואופרנדים).

Three general types of operators are available in Java:

1. **Arithmetic** operators (אופרטורים אריתמטיים)
2. **Relational** operators (אופרטורים יחס)
3. **Logical** operators (אופרטורים לוגיים)
Arithmetic operators perform arithmetic operations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>

Note: Operands’ type is very important.

5 / 2 yields an integer 2.
5.0 / 2 yields a double value 2.5
5 % 2 yields 1 (the remainder of the division)
To assist you with various types of calculations, Java contains a class named `Math`. In this class are the most commonly needed operations in mathematics.

<table>
<thead>
<tr>
<th>Method’s name</th>
<th>Operation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Returns the absolute value</td>
<td>Math.abs(-8)=8 Math.abs(15)=15</td>
</tr>
<tr>
<td>pow</td>
<td>Returns the value of the first argument raised to the power of the second argument.</td>
<td>Math.pow(2,3)=8.0 Math.pow(-4,2)=16.0</td>
</tr>
<tr>
<td>sqrt</td>
<td>Returns the positive square root of a double/integer value.</td>
<td>Math.sqrt(16)=4.0</td>
</tr>
<tr>
<td>round</td>
<td>Returns the closest to the argument.</td>
<td>Math.round(0.78)=1 Math.round(-8.2)=-8</td>
</tr>
</tbody>
</table>

Caution!

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.

*For example 1:*
```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```
displays `0.5000000000000001`, not `0.5`!

*For example 2:*
```
System.out.println(1.0 - 0.9);
```
displays `0.09999999999999998`, not `0.1`!

- Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.
Relational operators

All relational operators direct Java to return a Boolean value. Boolean value is either true or false.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>a &lt; 10</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>x1 &lt;= x2</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>num1 &gt; num2</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>b &gt;= c5</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>m3 == m4</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>x15 != 100</td>
</tr>
</tbody>
</table>
Logical operators are used to check if the results of relational expressions are true or false.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>!(x == y)</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(a1 == 10) &amp;&amp; (c &gt; 7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The value of the boolean expressions either **true** or **false**. The NOT operator **reverses** that value.

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
The result of logical AND is true if A and B are both true and false otherwise.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A &amp;&amp; B</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
The result of logical OR is true if A or B or both are true and false otherwise.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
<td>**A</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
A Java method body is a series of zero or more statements. There are many different kinds of statements in Java:

- Assignment statements
- Selection statements
- The switch statement
- The while statement
- The do-while statement
- The for statement
- The jump statement
- The return statement

One of the simplest is the Assignment Statement

<variable> = <expression>;}
Assignment Statements

Assignment statement changes the value stored in variable sides. A variable can store only one value of its declared type.

_**Java is strongly typed language.**_

*Casting* (המרת טיפוסים) is most general form of conversion types in Java.

```
x = 1;       // Assign 1 to x declared as integer
radius = 1.0; // Assign 1.0 to radius declared as double
a = 'A';    // Assign 'A' to a declared as char
```

*Casting examples:*

```
int x = 5, y = 2;
double a;
a = x/y;    ⇒ a = 2.0
a = (double)x/y; ⇒ a = 2.5
```

```
double b = 3.14;
int c;
c=(int)b;  ⇒ c = 3
```

Cast operator
# Shortcut Assignment Operators

variable = variable operator expression  \(\iff\) variable operator = = expression

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>f -= 8.0</td>
<td>f = f - 8.0</td>
</tr>
<tr>
<td>*=</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>%=</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>

```c
int x, y;
x*= y+5;  \iff\quad x = x*(y + 5);
x+= y;    \iff\quad x = x + y;
y-= 10;  \iff\quad y = y - 10;
```
## Increment and Decrement operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++var</td>
<td>preincrement</td>
<td>The expression (++var) increments var by 1 and evaluates to the <em>new</em> value in var after the increment.</td>
</tr>
<tr>
<td>var++</td>
<td>postincrement</td>
<td>The expression (var++) evaluates to the <em>original</em> value in var and increments var by 1.</td>
</tr>
<tr>
<td>--var</td>
<td>predecrement</td>
<td>The expression (--var) decrements var by 1 and evaluates to the <em>new</em> value in var after the decrement.</td>
</tr>
<tr>
<td>var--</td>
<td>postdecrement</td>
<td>The expression (var--) evaluates to the <em>original</em> value in var and decrements var by 1.</td>
</tr>
</tbody>
</table>

```plaintext
int x,y;
x=8;       x=8;       x=8;       x=8;
y=x++;    y=x++;    y=++x;    x=x+1;
  x=x+1;  y=x;       y=x;
yields y=8  yields y=9  yields x=9  yields j=38
  x=9      y=9        i=6         j=45
```

```plaintext
int i=5, j;
j=3+(7 * i++);  j=3+(7 * ++i);
yields j=38  yields i=6  yields j=45
```
import java.util.*;
public class Lec2Example
{
    static Scanner reader = new Scanner(System.in);
    public static void main(String[ ] args)
    {
        System.out.print("Enter the three digits number: ");
        int num = reader.nextInt();
        int a = num/100;
        int b = num%10;
        int c = num/10%10;
        num = 100*b + 10*c + a;
        System.out.println("The new value is: "+num);
    } //main
} //class

Enter the three digits number: 123
The new value is: ?
Selection statements

If it is raining....

- If it is raining: Take an umbrella
- If it is not raining: false
An *if-else statement* allows a program to do one thing if a logical expression is true and another thing otherwise.

```
if (expression)
    statement₁ (or block of statements)
else
    statement₂ (or block of statements)
```

- if *expression* is true, *statement₁* is executed.
- if *expression* is false, *statement₂* is executed.
- *statement* can be replaced by a block of statements, enclosed in curly braces:

```
if (expression)
{
    statements ... 
}
```
int gradeFinalExam = 60, gradeClassProject = 90;

if (gradeFinalExam >= 70 && gradeClassProject >= 80)
{
    System.out.println("Pass");
    gradeFinalExam += 10;
}  
else
{
    System.out.println("Fail");
    gradeFinalExam -= 5;
}
```java
boolean flag;
int gradeFinalExam = 60, gradeClassProject = 90;

flag = gradeFinalExam >= 70 && gradeClassProject >= 80;
if (flag)
{
    System.out.println("Pass");
    gradeFinalExam += 10;
}
else
    System.out.println("Fail");
```
nested if - else statement

grade >= 90

grade >= 80

grade >= 70

Yes

No

A

B

D

C

Yes

No

Yes

No

Yes

No
import java.util.*;
public class If_Else_Nested
{
    static Scanner reader = new Scanner(System.in);
    public static void main(String[] args)
    {
        int grade;//student's grade
        System.out.print("enter student's grade => ");
        grade = reader.nextInt();
        if(grade>=90)
            System.out.println("A");
        else
            if(grade>=80)
                System.out.println("B");
            else
                if(grade>=75)
                    System.out.println("C");
                else
                    System.out.println("D");
    }
}
The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. Note that value1, ..., and valueN are constant expressions, meaning that they cannot contain variables in the expression.

```java
switch (switch-expression) {
    case value1: statement(s)1;
               break;
    case value2: statement(s)2;
               break;
    ...
    case valueN: statement(s)N;
               break;
    default: statement(s);
} //switch
```
Switch statement example

```
switch (grade/10) {
    case 9:
        System.out.println("A");
        break;
    case 8:
        System.out.println("B");
        break;
    case 7:
        System.out.println("C");
        break;
    case 6:
        System.out.println("D");
        break;
    default:
        System.out.println("E");
} //switch
```

A `break` statement used to break out of each case of a switch.

If no case value matches that of the expression, execution continues with the optional `default`.