

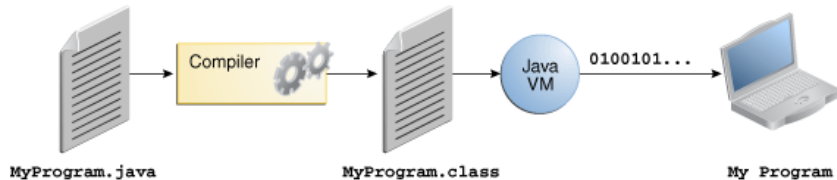
Programming Languages

Java Basics

Janyl Jumadinova

January 23 - 27, 2023

Java program development process



Simple first Java Program: “Hello World”

`/** This is the first program people write in a new language, the "Hello World!". In Java, this file must be named Welcome.java, with the first part of the name, Welcome, being the same as the name of the class. The filename itself (not the class name) must always end in .java to indicate to the operating system that it's a java source file.`

```
*/  
public class Welcome {  
    public static void main ( String args[] ) {  
        System.out.println ( "Hello World!" );  
    }  
}
```

Comments

Comments in Java can be one of three styles:

- **Single line:** starts at `//` anywhere on a line, ends at the end of that line
- **Multi-line:** starts with character sequence `/*` anywhere, ends with character sequence `*/` anywhere after that can span multiple lines
- **javadoc:** starts with character sequence `/**` anywhere, ends with character sequence `*/` anywhere, after that uses javadoc utility to create HTML documentation from code

- **public class Welcome:**

- **public** means that something is available across packages (reserved word)
- Name of the class has to be the same as the name of the .java file

- **public class Welcome:**
 - **public** means that something is available across packages (reserved word)
 - Name of the class has to be the same as the name of the .java file
- **public static void main (String identifier[]):**
 - The particular form of main is required by Java.
 - JVM starts executing here!
 - main is a static method, it is part of its class and not part of objects.
 - Strings in Java are sequence of characters

- **public class Welcome:**
 - **public** means that something is available across packages (reserved word)
 - Name of the class has to be the same as the name of the .java file
- **public static void main (String identifier[]):**
 - The particular form of main is required by Java.
 - JVM starts executing here!
 - main is a static method, it is part of its class and not part of objects.
 - Strings in Java are sequence of characters
- Braces { } are used to collect statements into a "block"

- **public class Welcome:**
 - **public** means that something is available across packages (reserved word)
 - Name of the class has to be the same as the name of the .java file
- **public static void main (String identifier[]):**
 - The particular form of main is required by Java.
 - JVM starts executing here!
 - main is a static method, it is part of its class and not part of objects.
 - Strings in Java are sequence of characters
- Braces { } are used to collect statements into a "block"
- Statements in Java end with semicolons.

Printing

- `println`: New line after printing
- `print`: No new line
- `printf`: Can specify format

Escape Sequences

- Escape sequences, or escape characters, begin with a slash and are immediately followed by another character.
- This two-character sequence, inside “ ” allows you to control your output (`\n`, `\t`, `\b`) or output characters you wouldn't otherwise be able to (`\\`, `\"`) inside a string.

Escape Sequences

Seq	Meaning	Example Code
<code>\n</code>	New line	<code>System.out.println(" Hi\nThere");</code>
<code>\t</code>	Horizontal tab	<code>System.out.println(" What's\tup?");</code>
<code>\b</code>	Backspace	<code>System.out.println(" Hi\b Hey");</code>
<code>\\</code>	Backslash	<code>System.out.println(" Back\\Slash");</code>
<code>\"</code>	Double quote	<code>System.out.println(" Dbl\" Quote");</code>

Variables

- **Variable** is a name for a memory's location where a data value is stored.

Variables

- **Variable** is a name for a memory's location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type
`int count;`

Variables

- **Variable** is a name for a memory's location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type
`int count;`
- **Variable Assignment** assigns a value to the variable
`count = 0;`

Variables

- **Variable** is a name for a memory's location where a data value is stored.
- **Variable Declaration** allows the compiler to reserve space in the main memory that is large enough for the specified type
`int count;`
- **Variable Assignment** assigns a value to the variable
`count = 0;`
- Must give a value to the variable before using it in the `main` method.

Java Identifiers

- reserved keywords (`class`, `public`, `static`, `void`)
- Java classes, methods, variables: words we chose or make up when writing a program
`System`, `println`, `main`, `args`

Java Identifiers

- reserved keywords (`class`, `public`, `static`, `void`)
- Java classes, methods, variables: words we chose or make up when writing a program
`System`, `println`, `main`, `args`

Identifier

a letter followed by zero or more letters (including \$ and _) and digits

Identifier Rules

- Identifiers must start with a letter, a currency character (\$), or a connecting character such as the underscore (_).
- Identifiers cannot start with a number.
- After the first character, identifiers can contain any combination of letters, currency characters, connecting characters, or numbers.
- There is no limit to the number of characters an identifier can contain.
- You can't use a Java keyword as an identifier.
- Identifiers in Java are case-sensitive; `foo` and `FOO` are two different identifiers.

Data Types

- Data stored in memory is a string of bits (0 or 1)

Data Types

- Data stored in memory is a string of bits (0 or 1)
- How the computer interprets the string of bits depends on the context.
- In Java, we must make the context explicit by specifying the type of the data.

Data Types

- Java has two categories of data:
 - primitive data (e.g., number, character)
 - object data (programmer created types)
- There are 8 primitive data types: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
- Primitive data are only single values; they have no special capabilities.

Primitive Data Types

- integers: `byte`, `short`, `int`, `long`
- floating point: `float`, `double`
- characters: `char`
- booleans: `boolean`

Common Primitive Data Types

Type	Description	Example of Literals
int	integers (whole numbers)	42, 60634, -8
double	real numbers	0.039, -10.2
char	single characters	'a', 'B', '&', '6'
boolean	logical values	true, false

Range of Values

Type	Storage	Range of Values
int	32 bits	-2,147,483,648 to 2,147,483,647
double	64 bits	$\pm 10^{-45}$ to $\pm 10^{38}$
char	16 bits = 2 bytes	0 to 2^{16} or \u0000 to \uFFFF
boolean	1 bit	NA

Constants

- Constants hold the same value during their existence.
- Can use a keyword `final` before the type and name of the variable:
 - always contains the same value.
- `final int MAX_BUDGET = 1000`

Scanner

- The `Scanner` class in the `java.util` package is a simple text scanner which can parse primitive types and strings.
- We can use the `Scanner` class to get the input.

Scanner

- The `Scanner` class in the `java.util` package is a simple text scanner which can parse primitive types and strings.
- We can use the `Scanner` class to get the input.
- We must first create an instance of the `Scanner` as:
`Scanner name = new Scanner (System.in);` to read from the terminal,

Scanner

- The `Scanner` class in the `java.util` package is a simple text scanner which can parse primitive types and strings.
- We can use the `Scanner` class to get the input.
- We must first create an instance of the `Scanner` as:
`Scanner name = new Scanner(System.in);` to read from the terminal,
or
`Scanner name = new Scanner(File filename);` to read from the file,
where
`File filename = new File("input.txt");`
and *name* is the name you choose for your instance of the `Scanner`

Scanner Methods

- `next()` : get the next word (token) as a `String`
- `nextLine()` : get a line of input as a `String`
- `nextInt()` : get an integer
- `nextDouble()` : get a double value
- `nextFloat()` : get a float value

Java API Packages

- The Java API (Application Programming Interface) contains many separate packages that can be used to make writing complex programs easier.
- Each package focuses on a specific set of tasks and provides pre-written methods :
 - `java.lang` - Fundamentals (Object, String, etc)
 - `java.util` - Utilities (Scanner , Random, etc)

String class

- `String str = 'abc';` is equivalent to:
`String str = new String('abc');`
- The `+` operator joins two strings together.

String class

- `String str = "abc";` is equivalent to:
`String str = new String("abc");`
- The `+` operator joins two strings together.
- `String` class is a part of the `java.lang` package.
- The classes of `java.lang` package are automatically available for use, no need to import.

String methods

charAt()

charAt() function returns the character located at the specified index.

```
String str = "studytonight";  
System.out.println(str.charAt(2));
```

Output : u

String methods

charAt()

charAt() function returns the character located at the specified index.

```
String str = "studytonight";  
System.out.println(str.charAt(2));
```

Output : u

length()

length() function returns the number of characters in a String.

```
String str = "Count me";  
System.out.println(str.length());
```

Output : 8

String methods

replace()

replace() method replaces occurrences of character with a specified new character.

```
String str = "Change me";  
System.out.println(str.replace('m','M'));
```

Output : Change Me

String methods

replace()

replace() method replaces occurrences of character with a specified new character.

```
String str = "Change me";  
System.out.println(str.replace('m','M'));
```

Output : Change Me

```
String str1 = new String("This is really fun!!");  
String str2 = str1.replace( 'i', 'u' );
```

String methods

replace()

replace() method replaces occurrences of character with a specified new character.

```
String str = "Change me";  
System.out.println(str.replace('m','M'));
```

Output : Change Me

```
String str1 = new String("This is really fun!!");  
String str2 = str1.replace( 'i', 'u' );
```

str2 returns "Thus us really fun"

String methods

substring()

substring() method returns a part of the string. **substring()** method has two forms,

```
public String substring(int begin);  
  
public String substring(int begin, int end);
```

The first argument represents the starting point of the substring. If the `substring()` method is called with only one argument, the substring returned, will contain characters from specified starting point to the end of original string.

But, if the call to `substring()` method has two arguments, the second argument specifies the end point of substring.

```
String str = "0123456789";  
System.out.println(str.substring(4));
```

Output : 456789

```
System.out.println(str.substring(4,7));
```

Output : 456

String methods

- `equals()`: This method returns true if the String are equal; false otherwise.

String methods

- **equals()**: This method returns true if the String are equal; false otherwise.

```
String str1 = new String("This is really fun!!");  
String str2 = new String("This is really fun!!");  
boolean value = str1.equals( str2 );
```


String methods

- **equals()**: This method returns true if the String are equal; false otherwise.

```
String str1 = new String('This is really fun!!');  
String str2 = new String('This is really fun!!');  
boolean value = str1.equals( str2 );
```

- value returns value = true

Control Structures

- Java programs are built from only these seven control structures:
 - *three selection* (if, if/else, switch)
 - *three repetition* (while, do/while, for)
- You implement computer algorithms by stringing sequences of these seven control structures together.

Logical Operators

- Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.
- These logical operators are based on the concept of Boolean logic or Boolean algebra.

Logical Operators

- Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.
- These logical operators are based on the concept of Boolean logic or Boolean algebra.
- These are the three logical operators in Java:
 - ① `&&` (logical AND)
 - ② `||` (logical OR)
 - ③ `!` (logical NOT, or negation)

Logical **and** Truth Table

<u>expr1</u>	<u>expr2</u>	<u>expr1 && expr2</u>
false	false	false
false	true	false
true	false	false
true	true	true

Logical **or** Truth Table

<u>expr1</u>	<u>expr2</u>	<u>expr1 expr2</u>
false	false	false
false	true	true
true	false	true
true	true	true

Logical **not** Truth Table

expr1

!expr1

false

true

true

false