



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering



Basic Concepts

Kaustubh Kulkarni
Assistant Professor,
Department of Computer Engineering,
KJSCE.

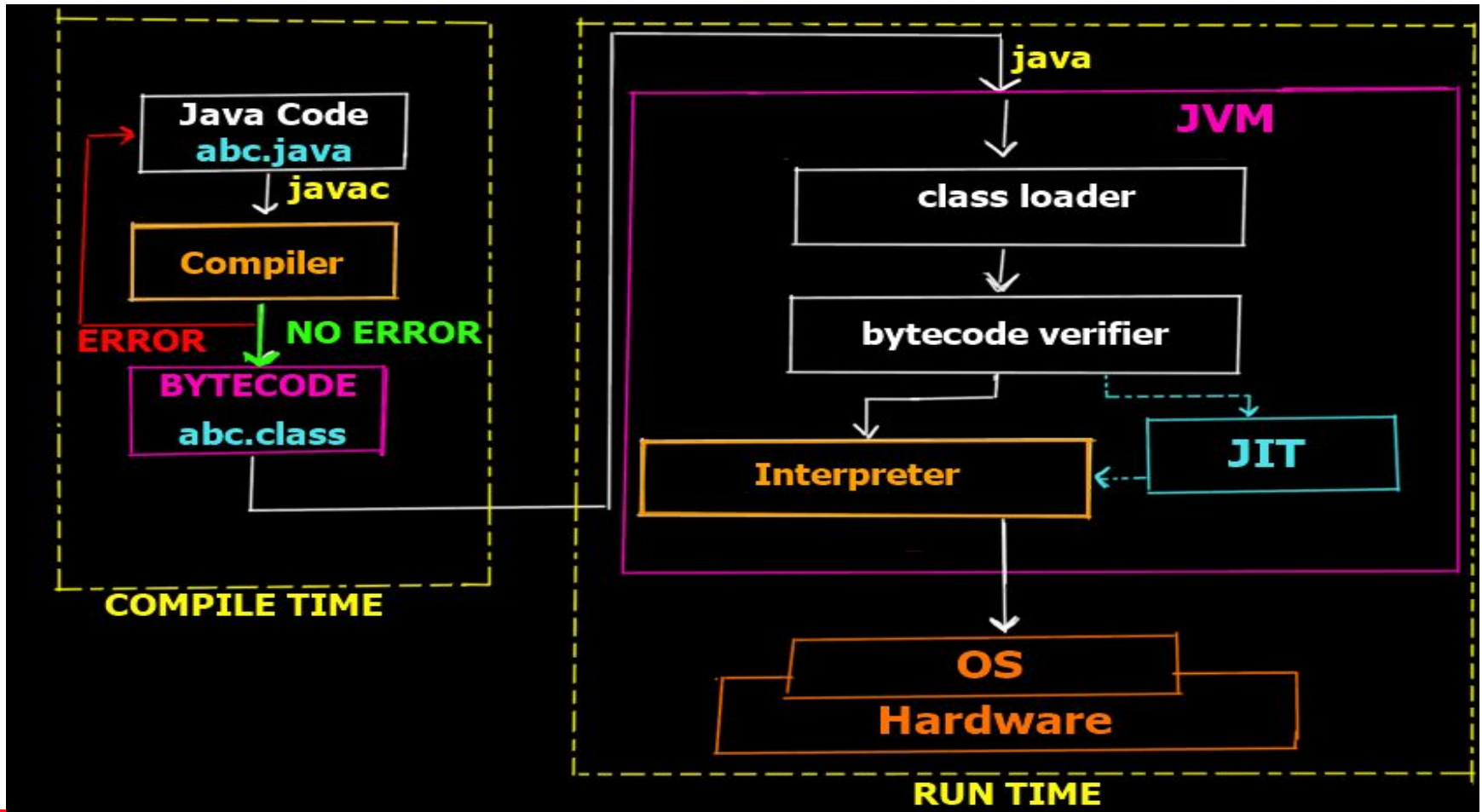
A Simple Program

```
/*  
This is a simple Java program.  
Call this file "Example.java".  
*/  
class Example {  
    // Your program begins with a call to main().  
    public static void main(String[] args) {  
        System.out.println("This is a simple Java program.");  
    }  
}
```

Creating the Source Code File

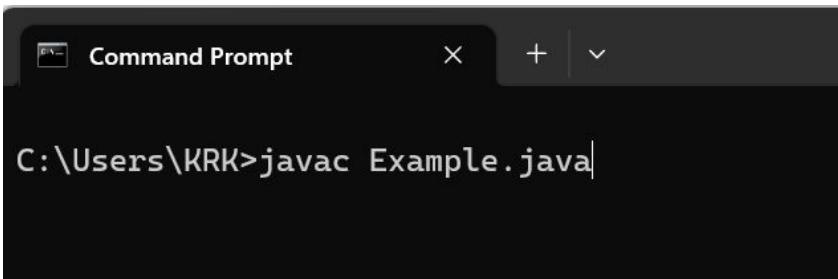
- For this example, the name of the source file should be **Example.java**.
- As you can see by looking at the program, the name of the class defined by the program is also **Example**.
- In Java, all code must reside inside a class.
- By convention, **the name of the main class should match the name of the file that holds the program.**

Compiling and Executing

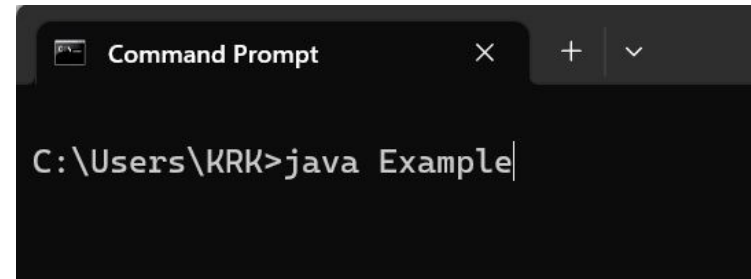


Compiling and Executing

- To compile the program, execute the compiler, **javac**, specifying the name of the source file on the command line
- The javac compiler creates a file called Example.class that contains the bytecode version of the program.
- To run the program, you must use the Java application launcher called **java** and pass the class name, Example, as a command-line argument



```
Command Prompt
C:\Users\KRR>javac Example.java
```

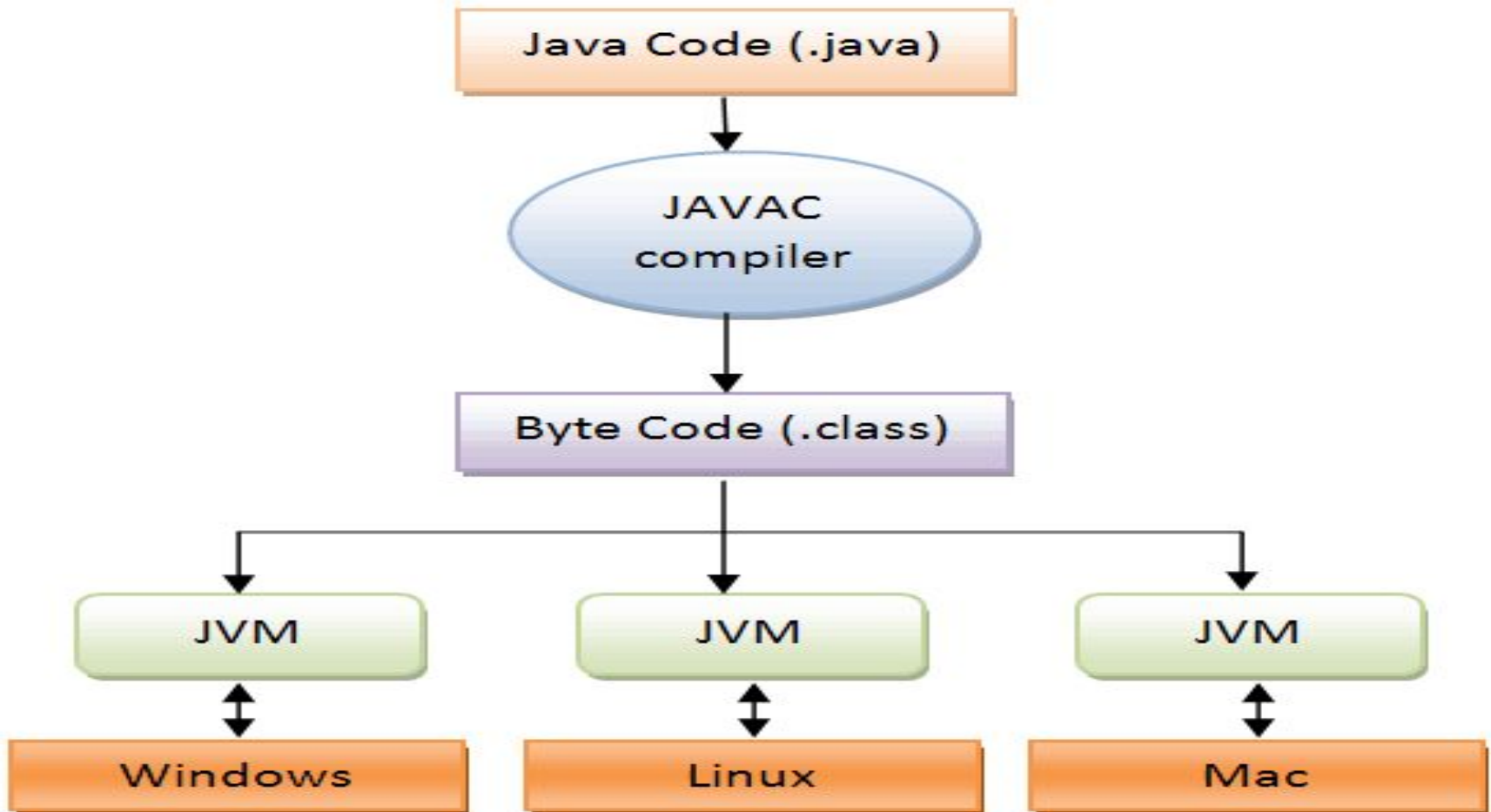


```
Command Prompt
C:\Users\KRR>java Example
```

Bytecode

- The Java bytecode is the intermediate representation of your program that contains instructions the **Java Virtual Machine (JVM)** will execute.
- This **intermediate code is platform independent** (you can take this bytecode from a machine running windows and use it in any other machine running Linux or MacOS etc).
- Also this bytecode is only understandable by the **JVM** and not the user or even the hardware /OS layer.

Java is Platform Independent



Run Time

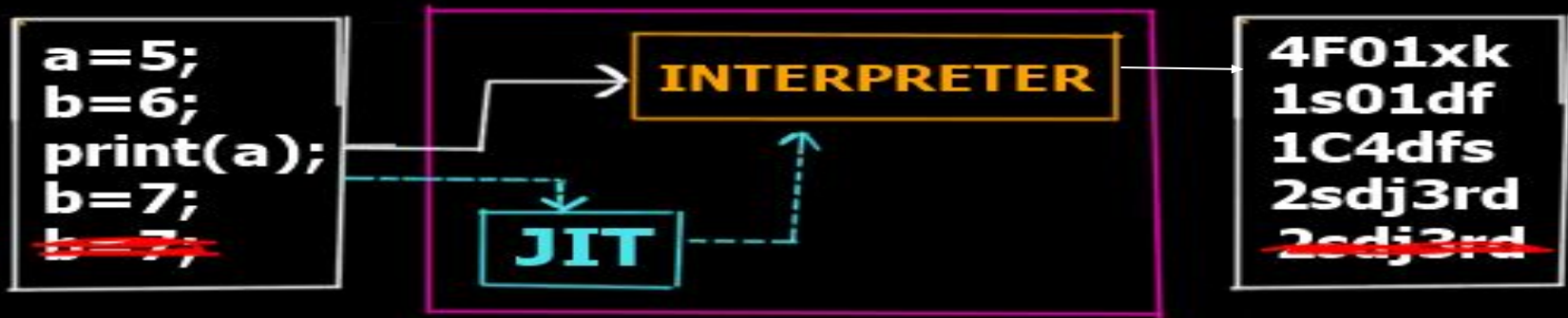
- Run Time phase starts when the bytecode is loaded into the JVM by the **class loader** (another inbuilt program inside the JVM).
- Now the **bytecode verifier** (an inbuilt program inside the JVM) checks the bytecode for its integrity and if no issues are found, passes it to the interpreter.
- The **interpreter** inside the JVM converts each line of the bytecode into executable machine code and passes it to the CPU to execute.

JIT (Just-In-Time) Compiler

CASE-1



CASE-2



JIT Compiler

- The last 2 lines (both $b = 7$) are the same (redundant).
- Clearly the second line does not have any effect on the actual output.
- Since the interpreter works line by line it still creates 5 lines of machine code for 5 lines of the bytecode in Case 1.
- This is inefficient.

JIT Compiler

- In case 2 we have the JIT compiler.
- Now before the bytecode is passed onto the interpreter for conversion to machine code, the **JIT compiler scans** the full code to see if it can be **optimized**.
- As it finds the last line is redundant it removes it from the bytecode and passes only 4 lines to the interpreter thus making it more **efficient** and **faster** as the interpreter now has 1 line less to interpret.
- JIT compiler optimizations can include inlining functions, constant folding, dead code elimination, and more. The goal is to produce highly efficient machine code tailored for the specific execution environment.



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering



A Closer Look at the Program

Multiline comment

The program begins with the following lines:

```
/*
```

```
This is a simple Java program.
```

```
Call this file "Example.java".
```

```
*/
```

```
This is a multiline comment.
```

Class Definition

- The next line of code in the program is :
class Example {
- This line uses the keyword **class** to declare that a new class is being defined.
- **Example** is an identifier that is the name of the class.
- The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).

Single-Line Comment

The next line in the program is the single-line comment, shown here:

```
// Your program begins with a call to main().
```

A single-line comment begins with a `//` and ends at the end of the line.

main()

- The next line of code is shown here:

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

- This line begins the main() method.
- This is the line at which the program will begin executing.
- As a general rule, a Java program begins execution by calling main().
- main() is simply a starting place for your program.
- A complex program will have dozens of classes, only one of which will need to have a main() method to get things started.

Printing output on screen

- The next line of code is shown here. Notice that it occurs inside `main()`.

`System.out.println("This is a simple Java program.");`

- This line outputs the string **"This is a simple Java program."** followed by a new line on the screen.
- This is accomplished by the built-in **`println()`** method.
- The line begins with **`System.out`**.
- **`System`** is a predefined class that provides access to the system, and **`out`** is the output stream that is connected to the console (screen).

Signature of main()

- The **public** keyword is an access modifier.
- When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.
- **main()** must be declared as **public**, since it must be called by code outside of its class when the program is started. (JVM)
- The keyword **static** allows **main()** to be called without having to instantiate a particular instance of the class.
- This is necessary since main() is called by the JVM before any objects are made.
- The keyword **void** simply tells the compiler that main() does not return a value.

Signature of main()

- **main()** has only one parameter **String[] args**
- **String[] args** declares a parameter named **args**, which is an array of instances / objects of the class **String**.
- Objects of type String store character strings.
- In this case, **args** receives any command-line arguments present when the program is executed.

Primitive Data Types

❖ **byte:**

- The **byte** data type is an 8-bit signed two's complement integer.
- It has a minimum value of -128 and a maximum value of 127 (inclusive).

❖ **short:**

- The **short** data type is a 16-bit signed two's complement integer.
- It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).

❖ **int:**

- By default, the **int** data type is a 32-bit signed two's complement integer.
- It has a minimum value of -2^{31} and a maximum value of $2^{31}-1$ (from -2147483648 to 2147483647, inclusive)

❖ **long:**

- The **long** data type is a 64-bit two's complement integer.
- The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$ (from -9223372036854775808 to 9223372036854775807, inclusive)

Primitive Data Types (continued)

❖ **float:**

- The float data type is a single-precision 32-bit IEEE 754 floating point.
- The maximum absolute value of a float in Java is 3.4028235 E 38 and the minimum absolute value is 1.4E-45.

❖ **double:**

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- The maximum absolute value of a double in Java is 1.7976931348623157 E 308 and the minimum absolute value is 4.9E-324.

❖ **Scientific notation:**

- **5 E 2 means $5 * 10^2$**

Primitive Data Types (continued)

- ❖ **boolean:**
 - The boolean data type has only two possible values: true and false.
 - Use this data type for simple flags that track true/false conditions.
 - This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- ❖ **char:**
 - The char data type is a single 16-bit Unicode character.
 - It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).
- ❖ In addition to the eight primitive data types listed till now, the Java programming language also provides special support for character strings via the `java.lang.String` class.
- ❖ Enclosing your character string within double quotes will automatically create a new `String` object.
- ❖ For example, `String s = "this is a string";`

Default Values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Program to demonstrate variables

```
class Example2 {  
    public static void main(String[] args) {  
        int num; // this declares a variable called num  
        num = 100; // this assigns num the value 100  
        System.out.println("You entered: " + num);  
        num = num *1.25 ;  
        System.out.print("Increased by 25 % :");  
        System.out.println(num);  
    }  
}
```


Explanation

- **int num; // this declares a variable called num**
- This line declares an integer variable called num.
- Java requires that variables be declared before they are used.
- Following is the general form of a variable declaration:
type var-name;
- Here, ***type*** specifies the type of variable being declared, and var-name is the name of the variable.

- **System.out.println("You entered: " + num);**
- In this statement, the plus sign causes the value of **num** to be appended to the string that precedes it, and then the resulting string is output.
- Actually, **num** is first converted from an integer into its string equivalent and then concatenated with the string that precedes it.

Input

- ❖ In Java, you can take input from the user using the **Scanner** class, which is part of the **java.util** package.
- ❖ **Step 1 : Import the Scanner class:**
 - `import java.util.Scanner;`
- ❖ **Step 2 : Create a Scanner object:**
 - `Scanner scanner = new Scanner(System.in);`
 - **System.in** represents the standard input stream, which is typically the keyboard.
- ❖ **Step 3 : Prompt the user for input:**
 - Although not strictly necessary, it's a good practice to provide a prompt to the user so they know what kind of input is expected. For example:
 - `System.out.print("Enter your name: ");`

Input (continued)

◆ Step 4 : Read input:

- The **Scanner** class provides various methods to read different data types.
- String `nextLine()`
 - Reads and returns the next line of input as a string. (Specifically, it reads and returns all the remaining characters on the line as a character string, and then moves to the next line.)
- String `next()`
 - Reads and returns the next input token as a character string.
- double `nextDouble()`
 - Reads and returns a double value. If the next token cannot be translated to a double, throws `InputMismatchException`.
- int `nextInt()`
 - Reads and returns an int value. If the next token cannot be translated to an int, throws `InputMismatchException`.
- In general, *type* `nextType()`
- For example:
 - `String name = scanner.nextLine();`
- Also refer the source code **ConsoleIO_Scanner.java**

Procedural Vs OOP

- All computer programs consist of two elements: code and data.
- Furthermore, a program can be conceptually organized around its code or around its data.
- That is, some programs are written around “what is happening” and others are written around “who is being affected.”
- Former, Procedural programming, latter, Object Oriented Programming.

Basic Terminology

- ❖ A *class* defines a blueprint for objects:
 - specifies the *attributes* (*data*) an object of the class can have.
 - provides *methods* specifying the *actions* an object of the class can take.
- ❖ An object is an *instance* of the class.
- ❖ Example: Person is a class. Alice and Bob are objects of the Person class.

Class Members

- ❖ *Members of a class:*
 - *Attributes (instance variables, data)*
 - For each instance of the class (object), values of attributes can vary, hence instance variables
 - *Methods*
- ❖ Example: class Person
 - Attributes: name, address, phone number
 - Methods: change_address(), change_phone_number()
- ❖ object Arthur
 - Name is Arthur, address is “221B Baker Street, London, England”, phone number is 123-4567
- ❖ object Bart
 - Name is Bart, address is “742 Evergreen Terrace, Springfield, USA”, phone number is 555-1212

State and Behavior

- An object's **state** refers to the current values of its attributes (also known as fields or instance variables).
- It represents the data associated with an instance of a class at a specific point in time.
- Each object of a class has its own unique set of attribute values, which collectively define its state.
- **Behavior** refers to the actions or operations that an object can perform.
- It defines what an object does and how it responds to various requests or stimuli.
- An object's behavior is defined by its methods.

Class Name: Automobile

Data:

amount of fuel _____

speed _____

license plate _____

Methods (actions):

increaseSpeed:

How: Press on gas pedal.

stop:

How: Press on brake pedal.

Class definition

Instantiations of the Class Automobile:

First Instantiation:

Object name: patsCar

amount of fuel: 10 gallons
speed: 55 miles per hour
license plate: "135 XJK"

Second Instantiation:

Object name: suesCar

amount of fuel: 14 gallons
speed: 0 miles per hour
license plate: "SUES CAR"

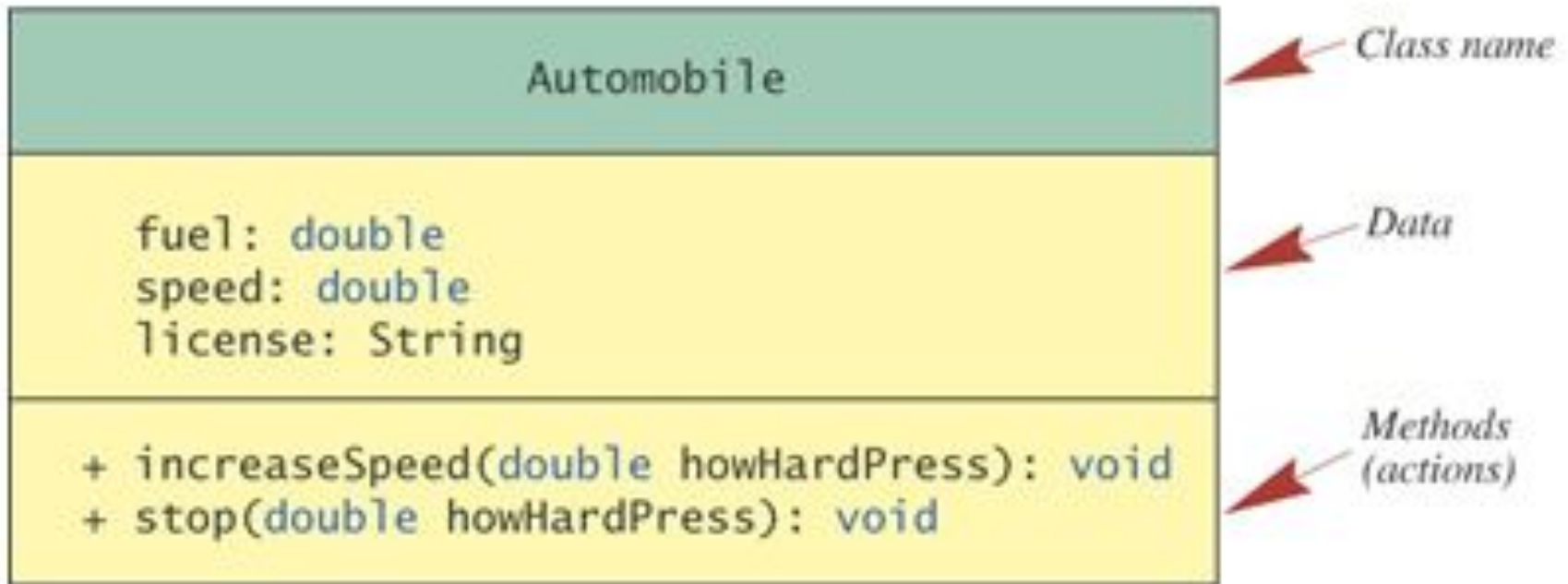
Third Instantiation:

Object name: ronsCar

amount of fuel: 2 gallons
speed: 75 miles per hour
license plate: "351 WLF"

*Objects that are
instantiations of the class*

A UML Class Diagram



- ❖ Declared within a method
 - “local to” (confined to) the method definition
 - can’t be accessed outside the method
- ❖ Not attributes (instance variables)

Naming Conventions

- **Class** names should be nouns, in mixed case, with the first letter of each internal word capitalized.
- Try to keep your class names simple and descriptive.
- Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).
- **Method** names should contain a verb, as they are used to make an object take action.
- They should be mixed case, beginning with a lowercase letter, and the first letter of each subsequent word should be capitalized.
- Adjectives and nouns may be included in method names:
 - `public void locate() {...} // verb`
 - `public String getBalance() {...} // verb and noun`
- **Instance and static variable** names should be nouns and should follow the same capitalization convention as method names:
 - `private String wayPoint;`

Naming Conventions

- **Parameter and local variable** names should be descriptive lowercase single words, acronyms, or abbreviations. If multiple words are necessary, they should follow the same capitalization convention as method names:

```
public void printHotSpots(ArrayList spotList) {  
    int counter = 0;  
    for (String hotSpot : spotList) {  
        System.out.println("Hot Spot #" +  
            ++counter + ": " + hotSpot);  
    }  
}
```

- **Temporary variable** names may be single letters such as i, j, k, m, and n for integers and c, d, and e for characters.
- **Constant** names should be all uppercase letters, and multiple words should be separated by underscores:

```
public static final int MAX_DEPTH = 200;
```

Creating Objects

- Creating objects of a class is a two-step process.
- **First**, you must declare a variable of the class type.
 - This variable does not define an object. Instead, it is simply a variable that can refer to (contain the address of) an object.
- **Second**, you must acquire an actual, physical copy of the object and assign it to that variable.
 - You can do this using the **new** operator.
 - The **new** operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.
 - This reference is, essentially, the address in memory of the object allocated by **new**.
- This reference is then stored in the variable.
- Thus, in Java, all the objects must be dynamically allocated.

Example : class Box

```
class Box {  
  
    double width;  
    double height;  
    double depth;  
  
}
```

```
class BoxDemo2 {
    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
        // assign values to mybox1's instance variables
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
        /* assign different values to mybox2's
        instance variables */
        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;
        // compute volume of first box
        vol = mybox1.width * mybox1.height * mybox1.depth;
        System.out.println("Volume is " + vol);
        // compute volume of second box
        vol = mybox2.width * mybox2.height * mybox2.depth;
        System.out.println("Volume is " + vol);
    }
}
```


Explanation

- ❖ In the preceding sample programs, a line similar to the following is used to declare an object of type Box:
 - `Box mybox = new Box();`
- ❖ It can be rewritten like this to show each step more clearly:
 - `Box mybox; // declare reference to object`
 - `mybox = new Box(); // allocate a Box object`
- ❖ The first line declares mybox as a reference to an object of type Box.
 - At this point, mybox does not yet refer to an actual object.
- ❖ The next line allocates an object and assigns its memory address to mybox.
- ❖ After the second line executes, you can use mybox as if it were a Box object.

Explanation (continued)

- ❖ Each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class.
- ❖ Thus, every Box object will contain its own copies of the instance variables **width**, **height**, and **depth**.
- ❖ To access these variables, you will use the **dot (.) operator**.
- ❖ The dot operator links the name of the object with the name of an instance variable.
- ❖ `mybox1.width = 10;`
- ❖ This statement tells the compiler to assign the value of 10 to the copy of width that is contained within the mybox1 object.
- ❖ In general, you use the dot operator to access both the instance variables and the methods within an object.

Illustration: Creating an object of type Box

Statement

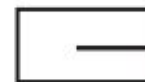
`Box mybox;`

Effect

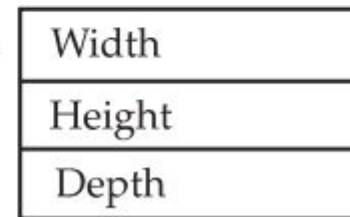


mybox

`mybox = new Box();`



mybox



Box object



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering



Questions?