Learn more about our research, discover data science, and find other great resources at:

http://www.dataminingapps.com

Chapter 2
Getting to Know Java

Overview

- A Short Java History
- Features Of Java
- Looking Under The Hood
- Java Language Structure
- Java Data Types

A Short Java History

- Sun Microsystems, Oak, 1991
- HotJava, first Java-enabled web browser in 1994
- Netscape incorporated Java in 1994
- Java 1.0 in 1995
- Open source in 1997 under GNU General Public License (GPL)
- In 2009, Sun was acquired by Oracle

A Short Java History

Major release	Date	Key characteristics
JDK 1.0	1996	First stable version of Java
JDK 1.1	1997	Inner classes; JavaBeans; JDBC; RMI; Just in Time (JIT) compiler for Windows platforms
J2SE 1.2	1998	Swing classes; Java IDL; Collections
J2SE 1.3	2000	Java platform debugger architecture (JPDA); JavaSound; HotSpot JVM
J2SE 1.4	2002	Regular expressions; IPv6 support; image I/O API; non-blocking I/O (nio); XML parser and XSLT processor
J2SE 5.0	2004	Generics; annotations; autoboxing; enumerations; varargs; for each loop
Java SE 6	2006	Improved GUI support; improved web service support
Java SE 7	2011	New file I/O capabilities; support for new network protocols
Java SE 8	2014	Lambda expressions; new date and time API
Java SE 9	2016 (expected)	Money and currency API

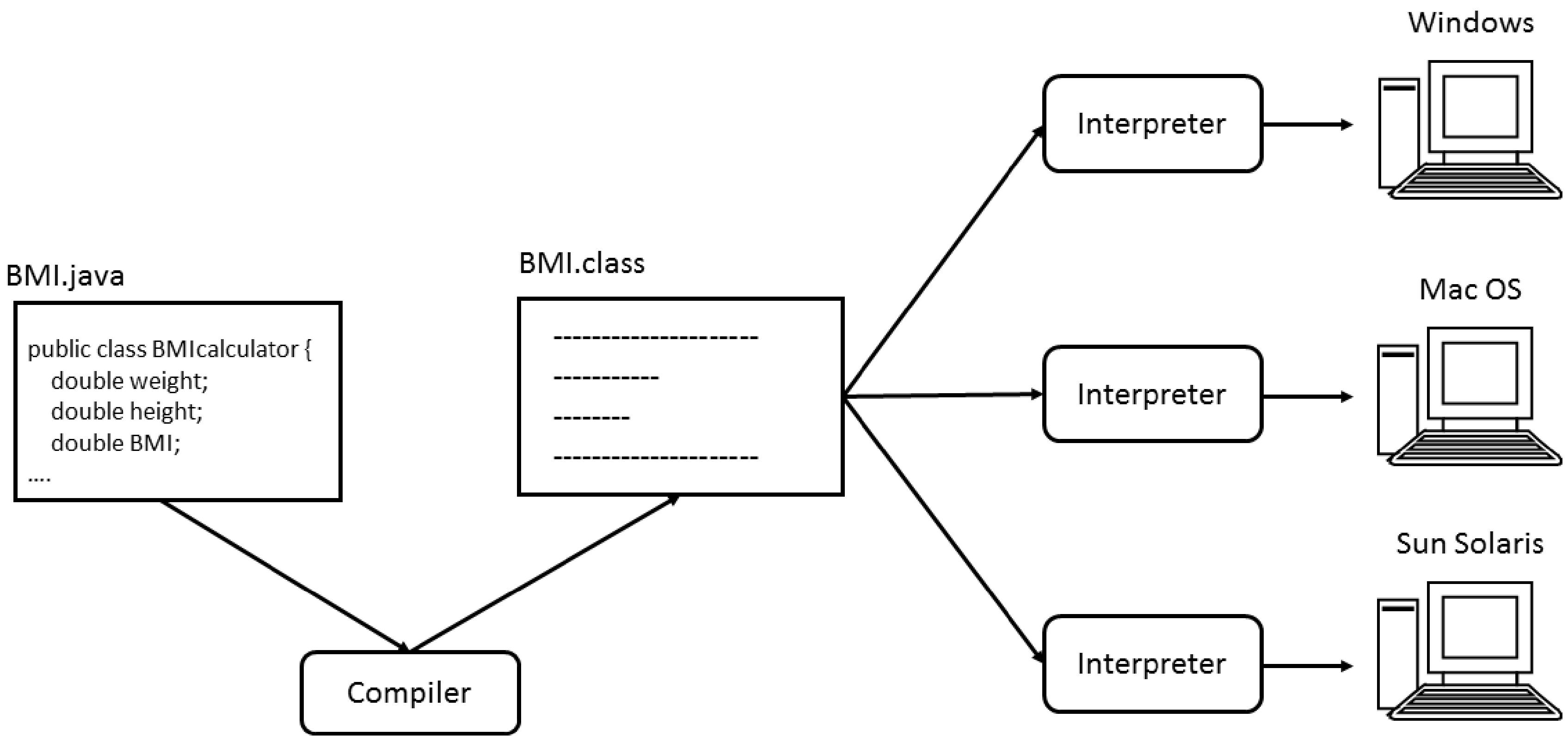
Features Of Java

- Simple
- Platform independent and portable
- Object Oriented (OO)
- Secure
- Multi-threaded
- Dynamic

Looking Under The Hood

- Bytecode
- Java Runtime Environment (JRE)
- Java Platforms
- Java Applications

Bytecode



```
public class BMIcalculator {
   double weight;
   double height;
   double BMI;
....
```

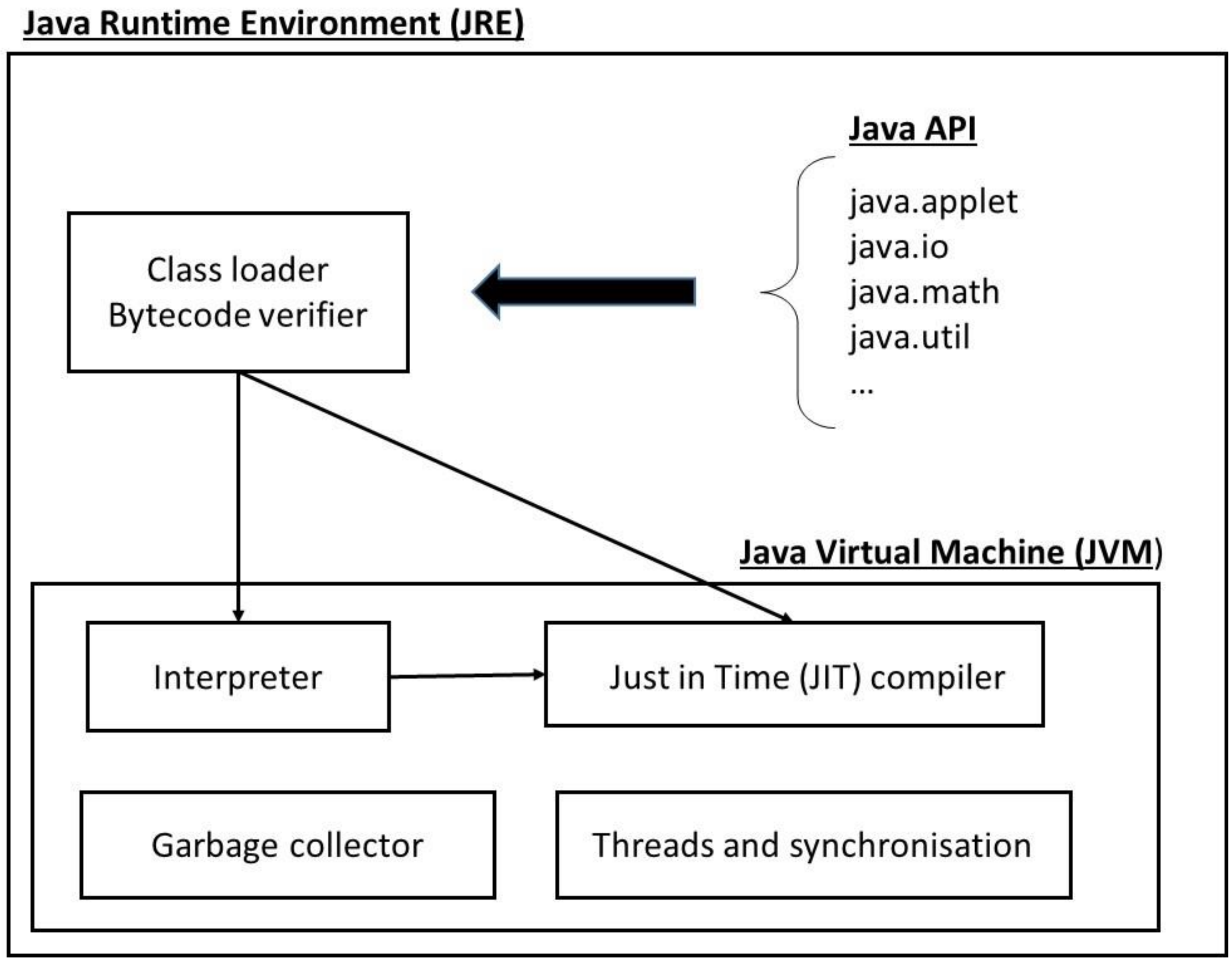
Bytecode

```
static double BMI;

 public BMIcalculator();
  Code:
    0: aload_0
    1: invokespecial #12         // Method java/lang/Object."<init>":()V
    4: return

 public static void main(java.lang.String[]);
  Code:
    0: ldc2_w    #20         // double 60.0d
    3: putstatic   #22       // Field weight:D
    6: ldc2_w    #24         // double 1.7d
    9: putstatic   #26       // Field height:D
   12: invokestatic #28      // Method calculateBMI:()V
   15: getstatic   #31       // Field java/lang/System.out:Ljava/io/PrintStream;
   18: new      #37          // class java/lang/StringBuilder
   21: dup
   22: ldc      #39          // String Your BMI is
   24: invokespecial #41     // Method java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
   27: getstatic   #44       // Field BMI:D
   30: invokevirtual #46     // Method java/lang/StringBuilder.append:(D)Ljava/lang/StringBuilder;
   33: ldc      #50          // String .
…
```

Java Runtime Environment (JRE)

- Java Application Programming Interface (API)
- Class Loader
- Bytecode Verifier
- Java Virtual Machine (JVM)

Java Runtime Environment (JRE)



Java Runtime Environment (JRE)
Java API
java.applet
java.io
java.math
java.util
...
Class loader
Bytecode verifier
Java Virtual Machine (JVM)
Interpreter
Just in Time (JIT) compiler
Garbage collector
Threads and synchronisation

Java Application Programming Interface (API)

Java Library	**Functionality**
`java.awt; javax.swing`	Support for creating graphical user interfaces (GUIs).
`java.applet`	Functionality to create applets.
`java.beans`	Functionality to create Java beans.
`java.io`	Support for I/O through files, keyboard, network, and so on.
`java.lang`	Functionality fundamental to the Java programming language.
`java.math`	Mathematical routines.
`java.security`	Security functions.
`java.sql`	Support for accessing relational databases by means of SQL.
`java.text`	Text support.
`java.util`	Various programming utilities.
`javax.imageIO`	Support for image I/O.
`javax.xml`	Support for XML handling.

Class Loader

- Locates and reads the *.class files and loads the bytecode into memory
- Classes are assembled into libraries stored as JAR files
- Bootstrap class loader: loads the core Java libraries
- Extensions class loader: loads the classes from the extensions directory
- System class loader: loads the code from the locations specified in the CLASSPATH environment variable

Bytecode Verifier

- Checks the validity of the bytecode
- Type checking all variables and expressions and ensures no unauthorized access to memory
- Can be disabled

Java Virtual Machine

- Abstract computer capable of excuting bytecode
- Write once, run everywhere philosophy
- E.g. HotSpot (Oracle)
- Components
  - Interpreter
  - Garbage collector
  - Multithreading and synchronization facilities
  - JIT compiler

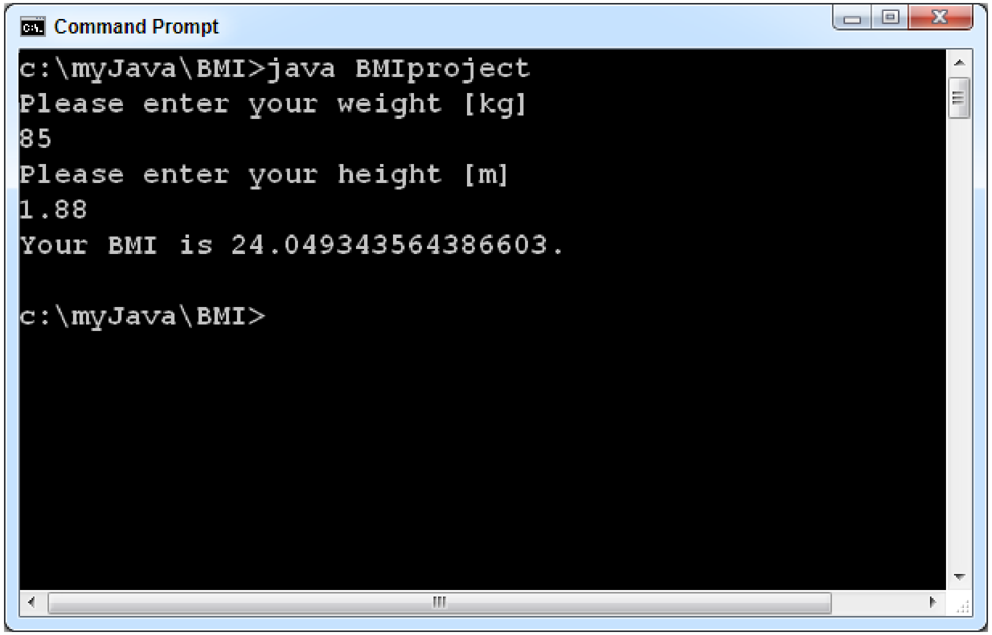
Java Platforms

Platform	**Key Characteristics**
J2SE (Java 2 Platform, Standard Edition)	Core Java platform designed for applications running on desktop PCs
J2EE (Java 2 Platform, Enterprise Edition)	Design, development, assembly, and deployment of business applications
J2ME (Java 2 Platform, Micro Edition)	Design of small, embedded applications in consumer devices (such as mobile phones)
Java Card	Design of small Java applications that run on smart cards
JavaFX	Design of Rich Internet Applications (RIAs)

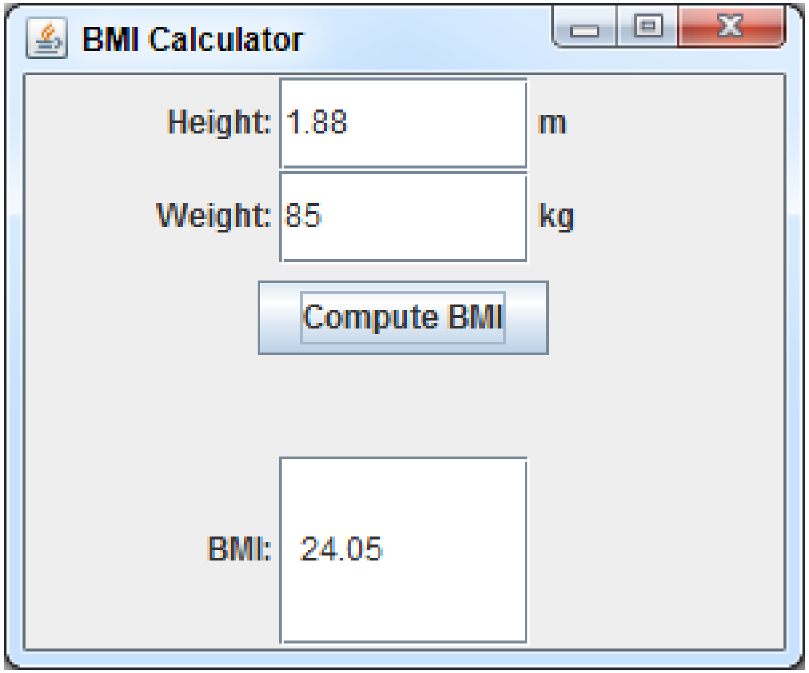
Java Applications

- Standalone Applications
- Java Applets
- Java Servlets
- Java Beans

Standalone Applications



```
c:\myJava\BMI>java BMIproject
Please enter your weight [kg]
85
Please enter your height [m]
1.88
Your BMI is 24.049343564386603.

c:\myJava\BMI>
```



Height: 1.88 m

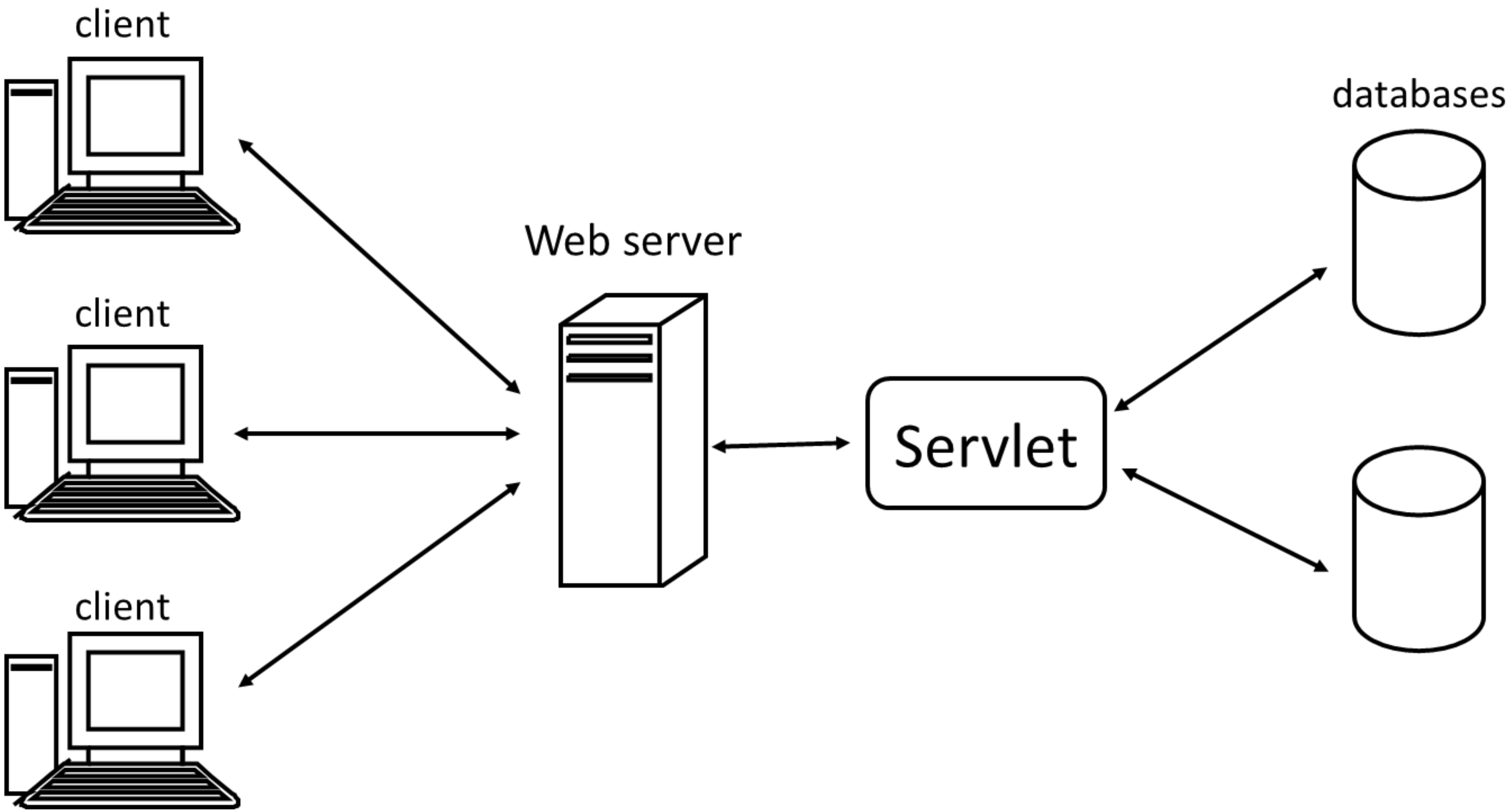
Weight: 85 kg

BMI: 24.05

Java Applets

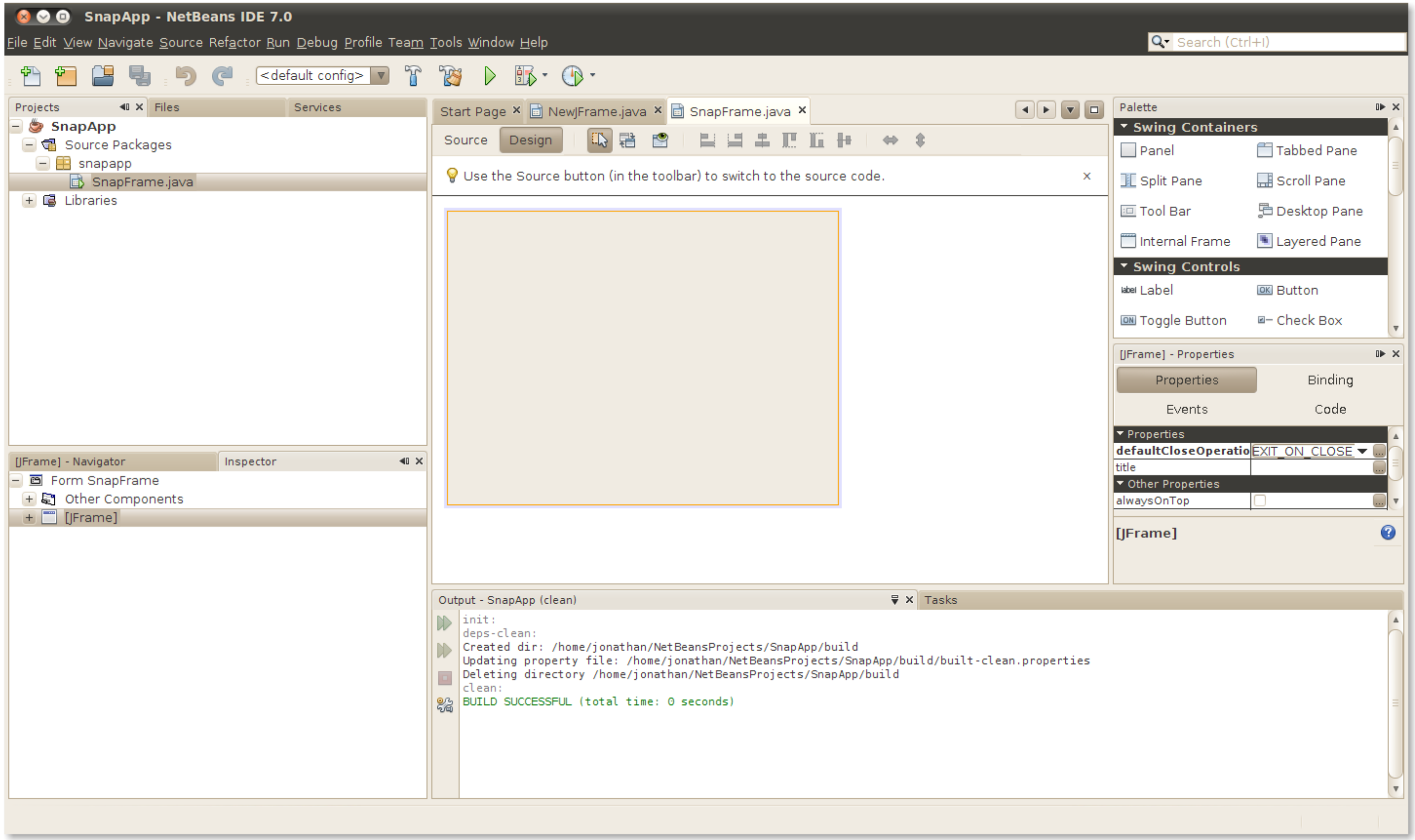
- Embedded in an HTML page using <Applet> … </Applet> tag
- Run in a sandbox
- Not popular anymore

Java Servlets



Java Beans

- Reusable software component that can be visually manipulated in a builder tool



Use the Source button (in the toolbar) to switch to the source code.

defaultCloseOperatio	EXIT_ON_CLOSE
title	
alwaysOnTop	□

```
init:
deps-clean:
Created dir: /home/jonathan/NetBeansProjects/SnapApp/build
Updating property file: /home/jonathan/NetBeansProjects/SnapApp/build/built-clean.properties
Deleting directory /home/jonathan/NetBeansProjects/SnapApp/build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```

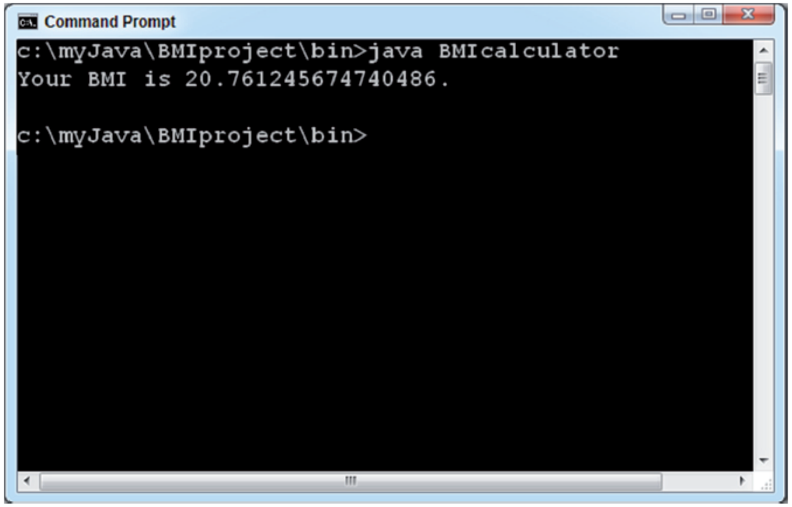
Java Language Structure

- Overview
- Classes
- Identifiers
- Java Keywords
- Variables
- Methods
- Comments
- Naming Conventions

Overview

```
public class BMICalculator {
// declare variables
double weight;
double height;
double bmi;

public BMICalculator(double w, double h) {
   weight = w;
   height = h;
}

public double calculateBMI() {
   return weight / (height * height);
}

// This is our main method.
public static void main(String[] args) {
   BMICalculator calculator = new BMICalculator(60, 1.70);
   double bmi = calculator.calculateBMI();
   // print BMI to screen
   System.out.println("Your BMI is " + bmi + ".");
}}
```

- Form-free language
- Statements end with ;



```
c:\myJava\BMIproject\bin>java BMIcalculator
Your BMI is 20.761245674740486.

c:\myJava\BMIproject\bin>
```

Classes

- Code container
- `public class BMICalculator{…}`
- Variables (e.g. `weight, heigth and BMI`) `and` methods (e.g. `BMICalculator, calculateBMI, main`)
- `Main` method is entry point of program execution

Identifiers

- Name of a language element
- E.g. class, variable or method
- E.g., `BMICalculator, weight, height`, `BMI`, `main` and `calculateBMI`
- Cannot begin with a digit or reserved keyword
- Java is case sensitive (e.g. `bmi`, `Bmi`, and `BMI` are all different!)

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Variables

- Name for a memory location that stores a specific value
- May change during program execution

```
// declare variables
double weight;
double height;
double BMI;
```

Methods

- Piece of code that performs a specific functionality
- Enclosed within bracktes { … }

```
public static void main(String[] args) {
   BMICalculator calculator = new BMICalculator(60, 1.70);
   double bmi = calculator.calculateBMI();
   // print BMI to screen
   System.out.println("Your BMI is " + bmi + ".");
}

public double calculateBMI() {
   return weight / (height * height);
}
```

Comments

- Improve code readability and facilitate maintenance
- Comments are not executed
- Examples
  - `// This is our main method.`

  -
    ```
    /* Here, we call the method calculateBMI which
     * will calculate the BMI.
    */
    ```
  - Javadoc tool produces HTML documentation from Java source code

Naming Conventions

Identifier	Convention	Good Examples	Bad Examples

Class	UpperCamelCase: The first letter of each word is capitalized	`BMICalculator`
		`Student`
		`MyProgram`
			`bmiCalculator`
			`STUDENT`
			`myProgram`

Variable	lowerCamelCase: The first letter is lowercase and the first letters of all following words are capitalized	`myHeight;`
		`myWeight;`
		`height;`
		`weight;`
			`MyHeight;`
			`myheight;`
			`Height;`
			`WEIGHT;`

Method	lowerCamelCase: The first letter is lowercase and the first letters of all following words are capitalized	`main` `calculateMyBMI`	`Main` `CalculateBMI`

Java Data Types

- Overview
- Primitive Data Types
- Literals
- Operators
- Arrays
- Type Casting

Overview

- Java is a strongly typed language
- Data type specifies how much memory to allocate to a variable, its format and its operations
- Primitive versus Composite data types

```
// declare variables
double weight;
double height;
double BMI;
```

Primitive Data Types

Type	Definition	Minimum	Maximum	Default
`byte`	8-bit signed integer	-128	127	0
`short`	16-bit signed integer	-32,768	32,767	0
`int`	32-bit signed integer	-2^{31}	$2^{31}-1$	0
`long`	64-bit signed integer	-2^{63}	$2^{63}-1$	0L
`float`	Single-precision 32-bit IEEE 754 floating point number	1.40239846x10^{-45}	3.40282347x10^{38}	0.0f
`double`	Double-precision 64-bit IEEE 754 floating point number	4.9406564581246544 x10^{-324}	1.79769313486231570 X10^{+308}	0.0d
`boolean`	One bit of information; flag indicator	false	true	false
`char`	Single 16-bit Unicode character	'\u0000' (or 0)	'\uffff' (or 65,535)	'\u0000'

Note: Java does not have a built-in string data type!

Literals

- Literal is a value assigned to a variable of a specific type
- Examples

```
weight = 60;
height = 1.70;
boolean overweight = true;
short age = 38;
double bmi = 24.2;
double bmi = 24.2d;
float bmi = 0.242e2;
char initial = 'B';
char gbPoundUniSymbol = '\u00A3';
char gbPoundSymbol = '£';
```

- Also possible to include escape characters (e.g. backspace, …)

Operators

- Perform data manipulations on one or more variables (operands)
- Types
  - Arithmetic operators
  - Assignment operators
  - Bitwise operators
  - Logical operators
  - Relational operators

Arithmetic Operators

Arithmetic Operator	**Example**	**Meaning**	**Result**
+	4+2	Addition	6
-	4-2	Subtraction	2
*	4*2	Multiplication	8
/	4/2	Division	2
%	8%3	Modulo (remainder after integer division)	2

Assignment Operators

Assignment Operator	Example	Meaning	Result
`=`	`weight = 85;`	Assign the value 85 to the variable weight	85
`+=`	`weight += 2;`	Same as `weight = weight + 2;`	87
`-=`	`weight -= 2;`	Same as `weight = weight - 2;`	85
`*=`	`weight *= 2;`	Same as `weight = weight * 2;`	170
`/=`	`weight /= 2;`	Same as `weight = weight / 2;`	85
`%=`	`weight %= 2;`	Same as `weight = weight % 2;`	1
`++`	`weight++;`	Same as `weight = weight + 1;`	2
`--`	`weight--;`	Same as `weight = weight - 1;`	1

Bitwise Operators

```
byte a = 40;     //binary a: 0010 1000
byte b = 122;    //binary b: 0111 1010
byte c = -12;    //binary c: 1111 0100
```

Bitwise Operator	Meaning	Examples	Result
&	Bitwise AND operator: Puts a 1 bit in the result if both input operands have a 1 bit at the given position.	a&b	a: 0010 1000 b: 0111 1010 r: 0010 1000
|	Bitwise OR operator: Puts a 1 bit in the result if one of both input operands have a 1 bit at the given position.	a|b	a: 0010 1000 b: 0111 1010 r: 0111 1010
^	Bitwise exclusive OR (XOR) operator: Puts a 1 bit in the result if one of the operands, but not both, has a 1 bit at the given position.	a^b	a: 0010 1000 b: 0111 1010 r: 0101 0010
~	Unary bitwise inverse operator: Changes every 1 bit to 0 and every 0 bit to 1.	~a	a: 0010 1000 r: 1101 0111

Bitwise Operators

```
byte a = 40;     //binary a: 0010 1000
byte b = 122;    //binary b: 0111 1010
byte c = -12;    //binary c: 1111 0100
```

Bitwise Operator	Meaning	Examples	Result
>>	Signed right shift operator: Shifts the left operand to the right by the number of bits specified. The left digits of a positive number are then filled with 0s, while the left digits of a negative number are filled with 1s. This preserves the original sign of the number, hence the name “signed right shift.”	a>>2 c>>2	a: 0010 1000 r: 0000 1010 c: 1111 0100 r: 1111 1101
>>>	Unsigned right shift operator: Shifts the left operand to the right by the specified number of bits. The left digits are always filled with 0s, regardless of the sign, hence the name “unsigned right shift.”	a>>>3 c>>>3	a: 0010 1000 r: 0000 0101 c: 1111 0100 r: 0001 1110

Bitwise Operators

```
byte a = 40;     //binary a: 0010 1000
byte b = 122;    //binary b: 0111 1010
byte c = -12;    //binary c: 1111 0100
```

Bitwise Operator	Meaning	Examples	Result
<<	Left shift operator: Shifts the left operand to the left by the number of bits indicated. The right digits are then filled with 0s. Since only the right side is filled, it is not possible to fill with 1s or 0s to ensure a positive or negative number. Therefore there is no distinction between a "signed left shift" and an "unsigned left shift."	a<<2 c<<2	a: 0010 1000 r: 1010 0000 c: 1111 0100 r: 1101 0000

Logical Operators

- Also known as boolean or conditional operators
- A: 3 > 2 (true)
- B: 2 < 1 (false)

Logical Operator	Meaning	Examples	Result
&&	Conditional AND operator: true if both operands are true.	A && B	false
||	Conditional OR operator: true if at least one operand is true.	A || B	true
^	Logical XOR operator: true if one, and only one, operand is true.	A ^ B	true
!	Unary NOT operator: true if the operand is false and vice versa.	!A	false

Logical Operator

Operand 1	Operand 2	AND	OR	XOR	NOT (Operand 1)
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Logical versus Bitwise And and OR operators

- AND
  - Bitwise: &; always evaluates both operands
  - Logical: &&; will not evaluate the second operand if the first one evaluates to false (short-circuiting behavior)
- OR
  - Bitwise: |; always evaluates both operands
  - Logical: ||; will not evaluate the second operand if the first one evaluates to true (short-circuiting behavior)

Relational Operators

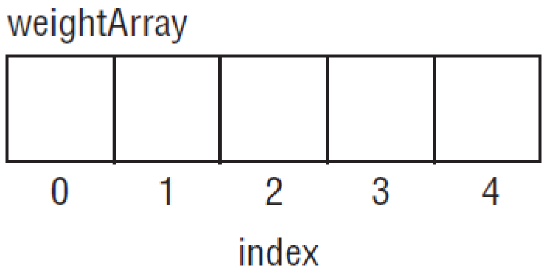
```
int a=4;
int b=9;
int c=4;
```

Relational Operator	Meaning	Examples	Result
`>`	Greater than: Verifies whether operand 1 is strictly bigger than operand 2.	`a > b`	`false`
`>=`	Greater than or equals: Verifies whether operand 1 is strictly bigger than or equal to operand 2.	`b >= a`	`true`
`<`	Less than: Verifies whether operand 1 is strictly lesser than operand 2.	`c < b`	`true`
`<=`	Less than or equals: Verifies whether operand 1 is strictly lesser than or equal to operand 2.	`b <= a`	`false`
`==`	Equal: Verifies whether operand 1 is equal to operand 2.	`a == c`	`true`
`!=`	Not equal: Verifies whether operand 1 is not equal to operand 2.	`a != b`	`true`

Arrays

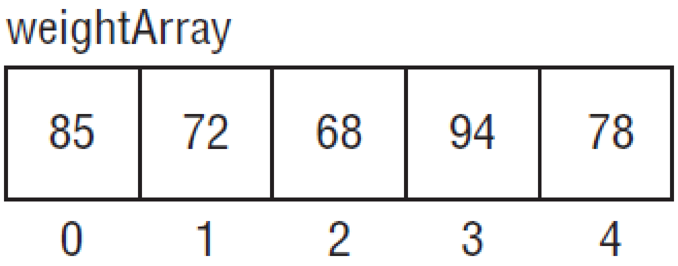
- Composite variable holding a fixed amount of values of a specific type (e.g. `int`, `long`, `char`, `float`, `double`, etc.)
- First element of the array has index 0
- Examples

```
float[] weightArray;
float weightArray[];
weightArray = new float[5];
```



Arrays

```
weightArray[0] = 85f;
weightArray[1] = 72f;
weightArray[2] = 68f;
weightArray[3] = 94f;
weightArray[4] = 78f;
```



```
float[] weightArray = {85f, 72f, 68f, 94f, 78f};
```

Arrays

```
public class Bmicalculator {

    // declare the three arrays
    static float[] weightarray;
    static float[] heightarray;
    static float[] BMIarray;

    // This is our main method.
    public static void main(String[] args){

        // initialize the three arrays as each having 5 elements

        weightArray = new float[5];
        heightArray = new float[5];
        BMIArray = new float[5];

        // assign the values to the weight array
        weightArray[0] = 85f;
        weightArray[1] = 72f;
        weightArray[2] = 68f;
        weightArray[3] = 94f;
        weightArray[4] = 78f;
```

Arrays

```
 //assign the values to the height array
        heightArray[0] = 1.74f;
        heightArray[1] = 1.80f;
        heightArray[2] = 1.90f;
        heightArray[3] = 1.84f;
        heightArray[4] = 1.88f;

        //compute the BMIs and store in the BMIArray
        BMIArray[0] = weightArray[0]/(heightArray[0]*heightArray[0]);
        BMIArray[1] = weightArray[1]/(heightArray[1]*heightArray[1]);
        BMIArray[2] = weightArray[2]/(heightArray[2]*heightArray[2]);
        BMIArray[3] = weightArray[3]/(heightArray[3]*heightArray[3]);
        BMIArray[4] = weightArray[4]/(heightArray[4]*heightArray[4]);

        // print the BMIs to the screen
        System.out.println("The BMI of person 1 is: " + BMIArray[0] + ".");
        System.out.println("The BMI of person 2 is: " + BMIArray[1] + ".");
        System.out.println("The BMI of person 3 is: " + BMIArray[2] + ".");
        System.out.println("The BMI of person 4 is: " + BMIArray[3] + ".");
        System.out.println("The BMI of person 5 is: " + BMIArray[4] + ".");
     }
}
```

Arrays

- Output gives:

```
The BMI of person 1 is: 28.075043.
The BMI of person 2 is: 22.222223.
The BMI of person 3 is: 18.836565.
The BMI of person 4 is: 27.76465.
The BMI of person 5 is: 22.06881.
```

Arrays

```
public class MatrixExample {

// This is our main method.
public static void main(String[] args){

   // declare and initialize the matrix
   int[][] matrix={{1, 2, 4},{2, 6, 8},{10, 20, 30}};

   // print some of the matrix numbers to the screen
    System.out.println("Element at row 0 and column 1 is: " + matrix[0][1] + .");
    System.out.println("Element at row 2 and column 2 is: " + matrix[2][2] + ".");
    System.out.println("Element at row 2 and column 1 is: " + matrix[2][1] + ".");
    System.out.println("Element at row 1 and column 0 is: " + matrix[1][0] + ".");
}}
```

Output is:

```
Element at row 0 and column 1 is: 2.
Element at row 2 and column 2 is: 30.
Element at row 2 and column 1 is: 20.
Element at row 1 and column 0 is: 2.
```

Arrays

```
public class MatrixExample {

    // declare and initialize the matrix
    static int[][] weirdMatrix={{1, 2},{2, 6, 8},{10}};

    // This is our main method.
    public static void main(String[] args){

        // print some of the matrix numbers to the screen
        System.out.println("Element at row 0 and column 1 is: " +
            weirdMatrix[0][1] + ".");
        System.out.println("Element at row 2 and column 2 is: " +
            weirdMatrix[2][0] + ".");
        System.out.println("Element at row 2 and column 1 is: " +
            weirdMatrix[1][2] + ".");
    }}
```

Output is:

```
Element at row 0 and column 1 is: 2.
Element at row 2 and column 0 is: 10.
Element at row 1 and column 2 is: 8.
```

Type Casting

- Converting a value from a specific type to a variable of another type
- Hierarchy of primitive data types (high to low precision): `double`, `float`, `long`, `int`, `short` and `byte`
- **Widening conversion (implicit casting):** value of narrower (lower precision) data type converted to value of a broader (higher precision) data type

```
int a = 4;
double x = a;
```

Or, explicitly:

```
int a = 4;
double x = (double) a;
```

Type Casting

- **Narrowing conversion (explicit casting):** value of broader (higher precision) data type converted to value of a narrower (lower precision) data type

  ```
  float b = 6.82f;
  int y = b;
  ```

  Following error will occur:

  ```
  Type mismatch: cannot convert from float to int
  ```

  Should be made explicit as follows:

  ```
  int y = (int) b;
  ```

Type Casting

```
public class TypeCastingExample {

    // declare and initialize the variables
    static int intA = 4;
    static float floatB = 6.82f;

    // This is our main method.
    public static void main(String[] args){

        //Widening conversion
        double doubleX = (double) intA;

        //Narrowing conversion
        int intY = (int) floatB;

        // print out the values
        System.out.println("The value of intA is: " + intA +".");
        System.out.println("The value of floatB is: " + floatB +".");
        System.out.println("The value of doubleX is: " + doubleX +".");
        System.out.println("The value of intY is: " + intY +".");
}}
```

Output:

```
The value of intA is: 4.
The value of floatB is: 6.82.
The value of doubleX is: 4.0.
The value of intY is: 6.
```

Type Casting

```
public class AnotherTypeCastingExample {

    public static void main(String[] args){

        float x = 3/9;
        float y = (float) 3/(float) 9;
        float z = (float) 3/9;

        System.out.println("The value of x is: " + x +".");
        System.out.println("The value of y is: " + y +".");
        System.out.println("The value of z is: " + z +".");
    }
}
```

Output:

```
The value of x is: 0.0.
The value of y is: 0.33333334.
The value of z is: 0.33333334.
```

Conclusions

- A Short Java History
- Features Of Java
- Looking Under The Hood
- Java Language Structure
- Java Data Types