



Chapter 1

# Building Blocks

# Learning about the environment

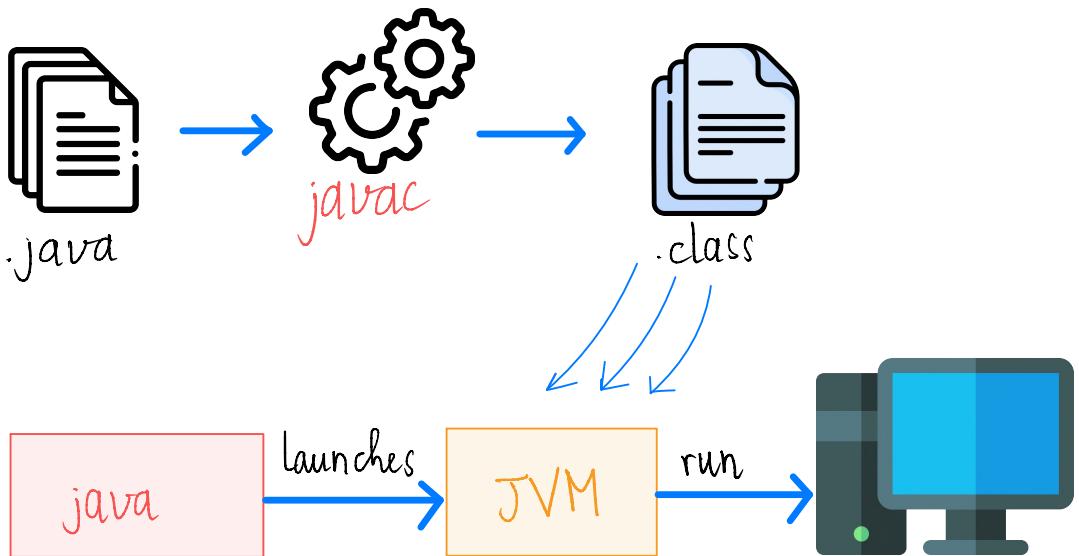


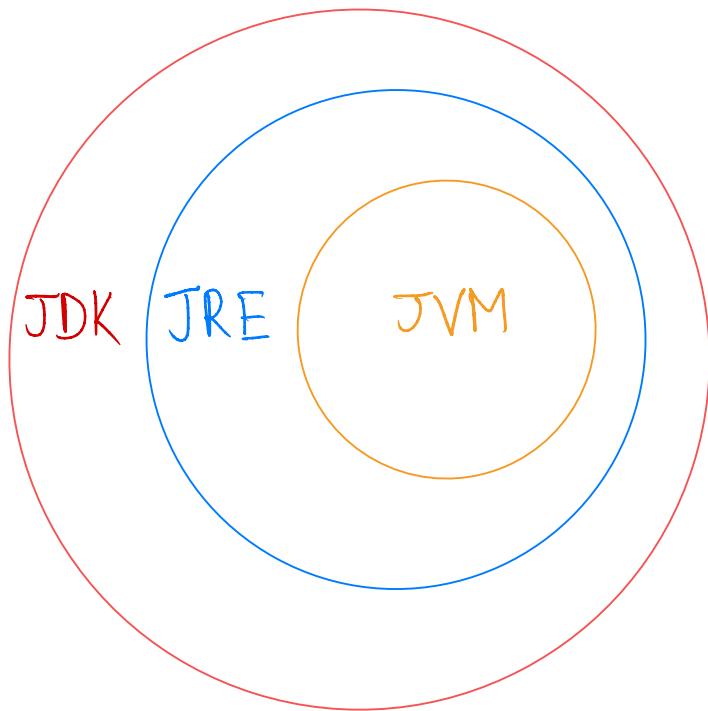
`javac`: converts .java source files  
→ .class bytecode

`java`: executes program

`jar`: packages files

`javadoc`: generates doc





Check version of Java:

`javac -version`

`java -version`

# Class structure

## Fields & methods

2 primary elements:

fields (variables) hold the state

methods (functions, procedures) operate on that state

method name + parameter types = method signature



```
public int numberVisitors (int month) {  
    return 10;  
}
```

→ signature: numberVisitors (int)

# Comments

1> // Single - line comment

2> /\* Multiple

    \*/  
    line comment

    don't have to

    \*/

3> /\*\*

    \* Javadoc multiple - line comment

    \* @author \_\_\_\_\_

    \*/

---



    \* /\* fish \*/



    \*/ → extra \*/ → error

# Classes & Source Files

public type needs to  
match  
file name

Animal.java

```
class Animal {  
    String name;  
}
```



```
public class Animal {  
    String name;  
}
```



```
public class Animal {  
    private String name;  
}  
  
class Animal2 {}
```



```
public class Animal {}  
public class Animal2 {}
```



AT MOST **1** of the top-level types  
in the file is allowed to be public

# Writing a main() Method

main() method: entry point into the program, the starting point that the JVM looks for when it begins running a new program

## Create main()

```
Zoo.java  
public class Zoo {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
        System.out.println(args[1]);  
    } //main() & args can be "final"  
}
```

required for main()

→ javac Zoo.java

can be:

→ java Zoo

String options[]

String... friends

# Passing Parameters

```
javac Zoo.java
```

```
java Zoo Bronx Zoo
```

↓      ↓  
args[0] args[1]

Output

Bronx  
Zoo

```
javac Zoo.java
```

```
java Zoo "San Diego" Zoo
```

↓      ↓  
args[0] args[1]

Output

San Diego  
Zoo

```
javac Zoo.java
```

```
java Zoo Zoo
```

↖

args[0]

Output

Zoo

Exception in thread "main" \_\_\_\_\_

args[1]

error: index out of bounds

```
javac __.java
```

```
java __ Args1
```

=

```
java __.java Args1
```

launching single-file source-code program

# Package declaration & import

## Packages

Package name begins with:

java → came with the JDK

com.wiley.javabook → associated with "wiley.com"

## Wildcards

import java.util.\*; → only one \*, must be at the end

import all classes in java.util

**BUT NOT** child packages, fields, or methods

java.lang (contains System class) is automatically imported

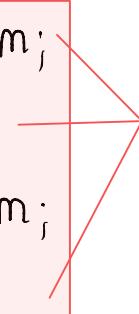
## Redundant Imports

import java.lang.System;

import java.lang.\*;

import java.util.Random;

import java.util.\*;

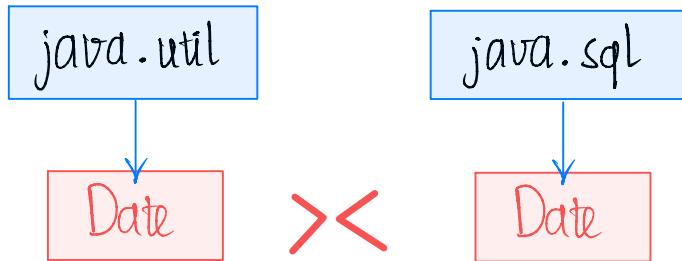


redundant

import java.nio.file.Path.\*;

cannot import methods

# Naming Conflicts



```
import java.util.*;  
import java.sql.*;
```

cause Date declaration  
to NOT compile

```
import java.util.Date;  
import java.sql.*;
```

✓  
default for Date

```
import java.util.Date;  
import java.sql.Date;
```

✗  
can't have 2 defaults

```
public class Conflicts {  
    java.util.Date date;  
    java.sql.Date sqlDate;  
}
```

✓  
fully qualified class name

# Creating a new package

```
package packagea;  
public class ClassA { }
```

```
package packageb;  
import packagea.ClassA;  
  
public class ClassB {  
    public static void main(String[] args) {  
        ClassA a;  
        System.out.println("Got it");  
    }  
}
```

\$ javac packagea/ClassA.java packageb/ClassB.java

\$ java packageb.ClassB

or: \$ javac packagea/\*.java packageb/\*.java

can't include subdirectories

packagea

  |  
  | Class A.java

  | Class A.class

packageb

  |  
  | Class B.java

  | Class B.class

) same dir

) same dir

default of javac

## Compiling to another directory

Important javac options

-cp <classpath>

Location of classes needed

or -classpath <classpath>

to compile the program

or --class-path <classpath>

-d <dir>

Directory in which to place  
generated class files

## Important java options

-cp <classpath>

or -classpath <classpath>

or --class-path <classpath>

↙ \$ javac -d classes packagea/ClassA.java  
packageb/ClassB.java

\$ java -cp classes packageb.ClassB

packagea

  └ ClassA.java

packageb

  └ ClassB.java

classes

  └ packagea

    └ ClassA.class

  └ packageb

    └ ClassB.class

Location of classes  
needed to run the program

# Compiling with JAR files



```
java -cp ".;C:\temp\otherLocation;C:\temp\myJar.jar"  
    myPackage.MyClass
```



```
java -cp ":/tmp/otherLocation:/tmp/myJar.jar"  
myPackage.MyClass
```

- → includes the current dir

`;` → separates parts of the classpath



```
java -cp "C:\temp\dirWithJars\*" myPackage.MyClass
```

add all JARs in dirWithJars

**BUT NOT** in subdirs of dirWithJars

# Creating a JAR file

## Important jar options

|                           |  |
|---------------------------|--|
| -c or --create            | creates a new JAR file                               |
| -v or --verbose           | prints details when working with JAR files           |
| -f (or --file) <fileName> | JAR filename   |
| -C <dir>                  | dir containing files to be used to<br>create the JAR |

 jar -cvf myNewFile.jar  
 jar --create --verbose --file myNewFile.jar  
 jar -cvf myNewFile.jar -C dir

## Ordering elements in a class

| Element                    | Example             | Required? |
|----------------------------|---------------------|-----------|
| Package declaration        | package abc;        | ✗         |
| import statements          | import java.util.*; | ✗         |
| Top-level type declaration | public class C      | ✓         |
| Field declarations         | int value;          | ✗         |
| Method declarations        | void method()       | ✗         |



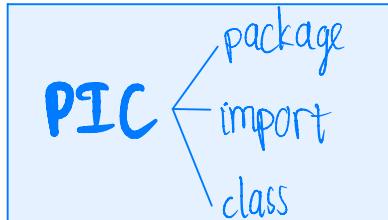
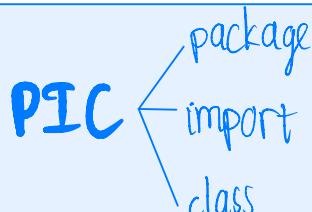
/\* header \*/

package structure;

// class Meerkat

public class Meerkat {}

 import java.util.\*;  
 package structure;  
 String name;  
 public class Meerkat {}

 PIC  
  
 package  
 import  
 class

# Creating Objects

## Calling Constructors

```
public class Chick {  
    public Chick() {  
        System.out.println("in constructor");  
    }  
}
```

→ match class's name

→ no return type

## Executing Instance Initializer Blocks

```
public class Bird {  
    public static void main(String[] args) {  
        System.out.println("Feathers");  
    }  
    {  
        System.out.println("Snowy");  
    }  
}
```

class definition

method declaration  
inner block

instance initializer (appear **OUTSIDE** a method)

# Following the Order of Initialization

- Fields & instance initializer blocks are run in the order in which they appear in the file
- The constructor runs after all fields & instance initializer blocks have run

```
public class Chick {  
    private String name = "Fluffy"; ②  
    { System.out.println("setting field"); } ③
```

```
public Chick() {  
    name = "Tiny"; ④  
    System.out.println("setting constructor");  
} ⑤
```

```
public static void main(String[] args) {  
    Chick chick = new Chick(); ①  
    System.out.println(chick.name); } } ⑥
```

# Output

setting field

setting constructor

Tiny

{ System.out.println(name); }

private String name = "Fluffy";

✗

# Understanding Data Types

## Primitive Types

the bit size depends on the JVM

| Keyword | Type                        | Min value      | Max value     | Default value | Example        |
|---------|-----------------------------|----------------|---------------|---------------|----------------|
| boolean | true or false               | n/a            | n/a           | false         | true           |
| byte    | 8-bit integral value        | -128           | 127           | 0             | 123            |
| short   | 16-bit integral value       | -32,768        | 32,767        | 0             | 123            |
| int     | 32-bit integral value       | -2,147,483,648 | 2,147,483,647 | 0             | 123            |
| long    | 64-bit integral value       | $-2^{63}$      | $2^{63} - 1$  | 0L            | 123L (or L)    |
| float   | 32-bit floating-point value | n/a            | n/a           | 0.0f          | 123.45f (or F) |
| double  | 64-bit floating-point value | n/a            | n/a           | 0.0           | 123.456        |
| char    | 16-bit Unicode value        | 0              | 65,535        | \u0000        | 'a'            |

(suffix won't be printed)

Octal (digits 0-7): 017

Hexadecimal (digits 0-9 + letters A-F/a-f):

0xFF, 0xff, 0XFF (case insensitive)

Binary (digits 0-1): 0b10, 0B10

---

Underscores

CAN'T be added at

the beginning / end of a literal

right before / after a decimal point

double notAtStart = -1000.00;

double notAtEnd = 1000.00\_-;

double notByDecimal = 1000\_-.00;



double annoyingButLegal = 1\_00\_0.0\_0;

double reallyUgly = 1\_\_\_\_\_2;



# Reference Types

A reference "points" to an object (an instance of a class) by storing the **memory address** where the object is located.

- A reference can be assigned to another object of the same or compatible type.
- A reference can be assigned to a new object using the "new" keyword.

| Primitive                           | Reference   |
|-------------------------------------|---|
| Lowercase type names                | Uppercase type names<br>(not required, standard only) |
| Don't have methods declared on them | Can be used to call methods<br>(when not null)        |
| Can NOT be assigned null            | Can be assigned null                                  |

# Wrapper Classes

| Primitive type | Wrapper class | Wrapper class inherits Number? | Example of creating        |
|----------------|---------------|--------------------------------|----------------------------|
| boolean        | Boolean       | No                             | Boolean.valueOf(true)      |
| byte           | Byte          | Yes                            | Byte.valueOf((byte) 1)     |
| short          | Short         | Yes                            | Short.valueOf((short) 1)   |
| int            | Integer       | Yes                            | Integer.valueOf(1)         |
| long           | Long          | Yes                            | Long.valueOf(1)            |
| float          | Float         | Yes                            | Float.valueOf((float) 1.0) |
| double         | Double        | Yes                            | Double.valueOf(1.0)        |
| char           | Character     | No                             | Character.valueOf('c')     |

```
Double apple = Double.valueOf ("200.99");
System.out.println(apple.byteValue()); // -56
System.out.println(apple.intValue()); // 200
System.out.println(apple.doubleValue()); // 200.99
```

valueOf() → reference type

parse\_() → primitive type

# Text Blocks

```
String textBlock = """  
    "Java Study Guide"  
    by Scott & Jeanne""";  
"""
```

Start text block      End text block  
Essential whitespace  
Incidental whitespace

make the code easier to read

Part of String

## Output

```
"Java Study Guide"  
by Scott & Jeanne
```

```
String pyramid = """
```

```
*
```

```
* *
```

```
* * *
```

```
"""
```

```
)
```

```
System.out.print(pyramid);
```

## Output

```
*
* *
* * *
(Empty Line)
```

4 lines

| Formatting             | Meaning in regular String  | Meaning in text block       |
|------------------------|--|-----------------------------|
| \\"\\\"\\\"            | "  | "                           |
| \\"\\\"\\\"\\\"        | n/a - Invalid  | ""                          |
| \\"\\\"\\\"\\\"\\\"    | ""   | ""                          |
| Space (at end of line) | Space  | Ignored                     |
| \s                     | Two spaces (\s is a space and preserves leading space on the line) | Two spaces                  |
| \n (at end of line)    | n/a - Invalid  | Omits new line on that line |

CS Scanned with CamScanner

String block = """" doe """;

no line break after the opening """"

String block =  
doe \\\\  
deer """;

doe deer → 1 line

String block =  
"doe\\\"\\\"\\\"  
\\\"deer\\\"\\\"\\\";  
";

System.out.print("\*" + block + "\*");

\* "doe"  
"deer"  
\*

# Declaring Variables

## Identifying Identifiers

An identifier is the name of a variable, method, class, interface, or package

- must begin with a letter / currency symbol (\$, ¥, etc.)/-symbol
- can include numbers
- a single - is NOT allowed
- can NOT use Java reserved words

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

\* The reserved words CONST and GOTO aren't actually used in Java. They are reserved so that people coming from other programming languages don't use them by accident—and, in theory, in case Java wants to use them one day.



```
long okIdentifier;  
float $OK2Identifier;  
boolean _alsoOKIdentifier;  
char __StillOKbutnotsonice$;
```

```
int 3DPointClass;  
byte hollywood@vine;  
String *$coffee;  
double public;  
short -i
```

(@), \* are not in  
the allowed list

## Declaring Multiple Variables

```
void paintFence() {  
    int i1, i2, i3 = 0;  
}
```

only i3 was initialized

```
int num, String value;
```

different types in the same statement

# Initializing Variables

## Creating Local Variables

Local variables: defined within a constructor, method, or initializer block

```
final int y = 10;  
int x = 20;  
y = x + 10; // Error
```

```
final int[] favoriteNumbers = new int[10];  
favoriteNumbers[0] = 10;  
favoriteNumbers[1] = 20;  
favoriteNumbers = null // Error
```

```
public int notValid() {  
    int y = 10;  
    int x; // not have a default value  
    int reply = x + y; // Error  
    return reply;  
}
```

```
public int valid() {  
    int y = 10;  
    int x;  
    x = 3;  
    int z;  
    int reply = x + y;  
    return reply;  
}
```

Local variables do not  
have a default value  
& must be initialized  
before use

# Passing Constructor & Method Parameters

```
public void findAnswer (boolean check) {}
```

```
public void checkAnswer () {  
    boolean value;  
    findAnswer (value); // DOES NOT COMPILE  
}
```

## Instance & Class Variables

- An **instance variable** (a field) is a value defined within a specific instance of an object.
  - A **class variable** is one that is defined on the **class level** & **shared** among all instances of the class.
- \* Don't need initializing
- \* Be given a default value as soon as we declare

# Inferring the Type with var

```
public class Zoo {  
    public void whatTypeAmI() {  
        var name = "Hello";  
        var size = 7;  
    }  
}
```

Only used for  
**LOCAL** variables  
Not used in a  
multi-variable  
assignment

```
public void reassignment() {  
    var number = 7;  
    number = 4;  
    number = "five"; // Error  
}
```

the type is set once at  
compile time & does not  
change

```
public void doesThisCompile(boolean check) {  
    var question; // Error  
    question = 1;  
    var answer; // Error  
    if (check) {  
        answer = 2;  
    } else {  
        answer = 3;  
    }  
    System.out.println(answer);  
}
```

question & answer are not assigned values on the lines where they are defined  
⇒ the compiler doesn't know what to make of them

```
public void twoTypes() {  
    int a, var b=3; //Error  
    var n=null; // Error  
}
```

NULL is not allowed  
for var  
(null could be any  
reference type)

```
public int add(var a, var b) {  
    return a+b;  
}
```

not used for  
method params

```
public class Var {  
    public void var() {  
        var var = "var";  
    }  
  
    public void Var() {  
        Var var = new Var();  
    }  
}
```

var is not a reserved word  
& allowed to be used as an  
identifier  
var is considered a **TYPE NAME** (cannot be used  
to define a **type**, such as a class,  
interface, or enum)

# Managing Variable Scope

## Limiting Scope

```
public void eatIfHungry(boolean hungry) {  
    if (hungry) {  
        int bitesOfCheese = 1;  
    }  
    var teenyBit = true;  
    System.out.println(bitesOfCheese);  
}  
System.out.println(teenyBit); //Error  
}
```

# Applying Scope to Classes

```
public class Mouse {  
    final static int MAX_LENGTH = 5;  
    int length;  
    public void grow(int inches) {  
        if (length < MAX_LENGTH) {  
            int newSize = length + inches;  
            length = newSize;  
        }  
    }  
}
```

stays in scope until  
the program ends

stays in scope  
as long as this  
Mouse object  
exists

# Destroying Objects

## Garbage Collection

refers to a process of automatically freeing memory on the heap by deleting objects that are no longer reachable in your program

eligible for garbage collection refers to an object's state of no longer being accessible in a program & therefore able to be garbage collected

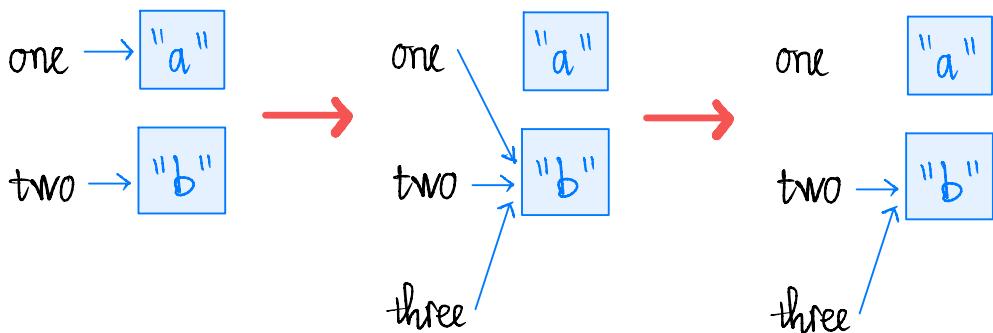
## Tracing Eligibility

An object will remain on the heap until it is no longer reachable:

- The object no longer has any references pointing to it.
- All references to the object have gone out of scope.

Garbage collection is **NEVER** guaranteed to run

```
public class Scope {  
    public static void main (String [] args) {  
        String one, two;  
        one = new String ("a");  
        two = new String ("b");  
        one = two; // "a" is eligible for g.c.  
        String three = one;  
        one = null;  
    }  
}
```



# Review Questions

- 1/D E ✓
- 2/C D E ✓
- 3/A E ✓
- 4/B E G ✓
- 5/A D F ✓
- 6/F ✓
- 7/C E ✓
- 8/B D H E (I/O is runtime exc.)
- 9/E ✓
- 10/A E F ✓

69,6%

- 11/D ~~E~~ (in the same package)
- 12/A C D ✓
- 13/A B C ✓
- 14/A B D E ✓
- 15/C E F ✓
- 16/A D ~~F~~ (incidental indentation)
- 17/D F ~~H~~ G (not print f suffix)
- 18/B C F ✓
- 19/A D (parseLong → primitive)
- 20/~~H~~C (long age; age = 1200; ok)
- 21/D ✓
- 22/C F G ✓
- 23/D ~~H~~ A (double doesn't fit float)