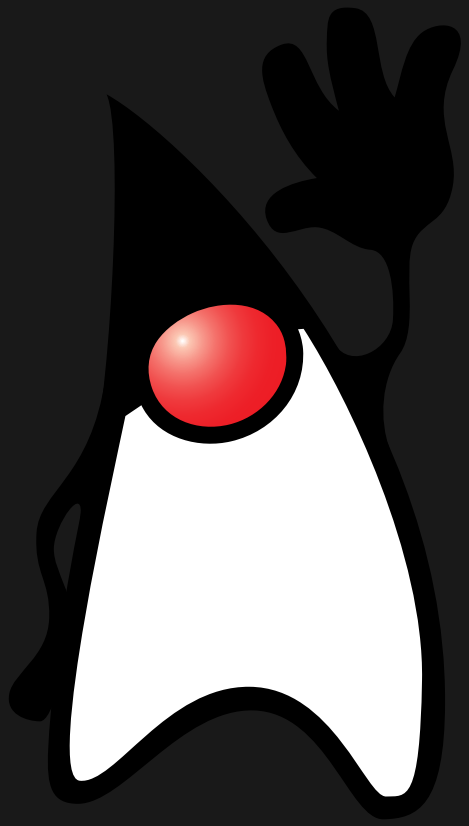
JAVA FOR QA



AGENDA

- Quick intro
- Hello QA
- Build tools & project template
- Language basics
  - primitive types
  - arrays
  - strings
  - operators
  - flow control

JAVA

JAVA

- 25 years old (!)
- object-oriented
- compiled to bytecode, run on JVM
- "write once run anywhere"
- static typing

```
String s = "blabla";
s = 123; // Error: incompatible type
```

- current version: 14 (11 - LTS)

HELLO QA

```
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello, QA!");
    }

}
```

```
$ javac Hello.java

$ java Hello
```

```
$ javac -d classes src/Hello.java

$ java -classpath classes Hello
```

BUILD TOOLS

Automate the creation of executable applications

- compilation
- dependencies management
- running tests
- packaging
- ...

MAVEN VS GRADLE

- wrappers
  - Maven: `./mvnw clean build`
  - Gradle: `./gradlew clean build`

GRADLE DEMO

```
$ mkdir gradle-demo && cd gradle-demo

$ gradle init
```

JAVA BASICS

PRIMITIVE VS REFERENCE TYPES

- primitive variables store the **actual value**
- reference variables store the **addresses of the objects**

```
int a = 77;
Person person = new Person();
```

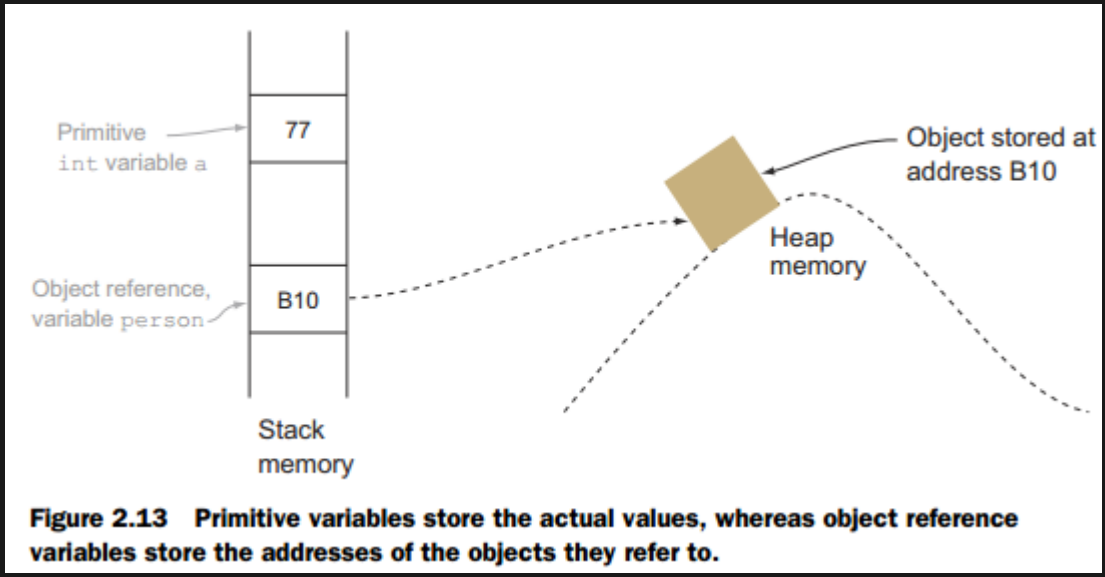


Figure 2.13 Primitive variables store the actual values, whereas object reference variables store the addresses of the objects they refer to.

PRIMITIVE DATA TYPES

Type	Description	Default value	Range	Examples
byte	8-bit signed integer	0	-128 .. 127	-2, -1, 0, 1, 2
short	16-bit signed integer	0	-32,768 .. 32,767	-2, -1, 0, 1, 2
int	32-bit signed integer	0	-2^{31} .. $2^{31}-1$	-2, -1, 0, 1, 2
long	64-bit integer	0L	-2^{63} .. $2^{63}-1$ (0 .. $2^{64}-1$)	-2L, -1L, 0L, 1L, 2L
float	32-bit IEEE 754 floating point	0.0f	-	3.14f, 1.23e100f
double	64-bit IEEE 754 floating point	0.0d	-	3.14d, 1.23e100d
char	16-bit Unicode character	'\u0000'	'\u0000' .. '\uffff' (0 .. 65,535)	'a', cha, '\101', '\n', 'ß'
boolean	true/false flag	false	true, false	true, false

GOOD PRACTISE

Avoid `float` and `double` if exact answers are required.

MOSTLY USED

- `int/long`
- `float/double`
- `boolean`

OPERATORS

ARITHMETIC OPERATORS

- +
- -
- *
- %

EQUALITY/RELATIONAL OPERATORS

- ==
- !=
- >
- >=
- <
- <=
- &&
- ||

TERNARY OPERATOR

booleanExpression ? expression1 : expression2

```
number > 0 ? "positive" : "not positive"
```

GOOD PRACTISE

Don't overuse ternary operator.

Good:

```
int a = (b > 10) ? c : d;
```

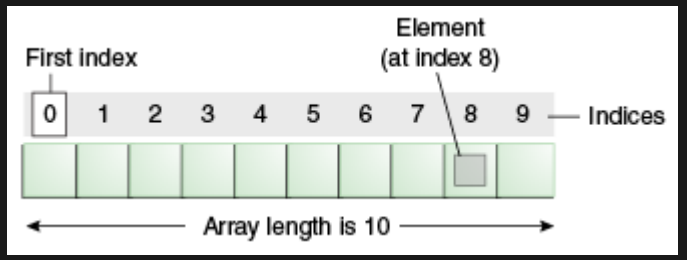
Bad:

```
int a = b > 10 ? c < 20 ? 50 : 80 : e == 2 ? 4 : 8;
```

REFERENCE TYPES

ARRAYS

- objects
- holds values of a single type
- fixed length
- indexed from 0



ARRAYS

```java
// array declaration
int[] arrayOfInts;
char[] arrayOfChars;

// initialization
arrayOfInts = new int[10]; // fixed size
int[] anotherArray = new int[5];
int[] array = {10, 20, 30, 40, 50};

// misc
System.out.println("Array length: " + array.length);
System.out.println("First element: " + array[0]);
System.out.println("Last element: " + array[array.length - 1])
array[0] = 111;
System.out.println(array[0]);
```

ARRAY MANIPULATIONS

- `java.util.Arrays` class provides several methods for array manipulations

```
char[] hello = {'h', 'e', 'l', 'l',  'o', '!'};
char[] copy = java.util.Arrays.copyOfRange(hello, 1, 5);
System.out.println(new String(copy));
```

STRINGS

- sequence of characters
- in Java strings are objects
- immutable
  - once created a String object cannot be changed

STRINGS

```
String hello = "Hello!";

// is equivalent to
char[] helloArray = { 'H', 'e', 'l', 'l', 'o', '!' };
String helloString = new String(helloArray);

// String Length
int len = helloString.length();
```

COMPARING STRINGS

```
String first = "JavaForQA";
String second = "JavaForQA";
System.out.println(first == second); // ???

String third = new String("JavaForQA");
String fourth = new String("JavaForQA");
System.out.println(third == fourth); // ???

String fifth = "JavaForQA";
String sixth = new String("JavaForQA");
System.out.println(fifth == sixth); // ???

// using equals
System.out.println(fifth.equals(sixth)); // ???
System.out.println(fifth.equals("javaforqa")); // ???
```

GOOD PRACTISE

- Use string literals instead of calling `new`.

Do:

```
String str = "abc";
```

Don't:

```
String str = new String("abc");
```

- Don't use == operator to compare Strings

Do:

```
str1.equals(str2)
```

Don't:

```
str1 == str2
```

CONTROL FLOW

- if-then, if-then-else
- while, do-while
- for
- switch
- break, continue, return
- statement vs expression

IF-THEN

```
void applyBrakes() {
    // bicycle must be moving
    if (isMoving){
        currentSpeed--;
    }
}

// with only one statement braces ({}) can be omitted
// but it can be error-prone:
void applyBrakes() {
    if (isMoving)
        currentSpeed--;
}
```

IF-THEN-ELSE

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stoppe
    }
}

char grade;
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
```

SWITCH

- works with integers, characters, Strings, Enums
- pay attention to `breake;`

```
String message;
switch (grade) {
  case 'A':
    message = "You're so smart!";
    break;
  case 'B':
    message = "Very good!";
    break;
  case 'C':
    message = "Not so bad!";
    break;
  default:
    message = "You're such a dumbass!";
    break;
}
```

WHILE

```
while (expression) {
    statement(s)
}
```

```
int count = 1;
while (count < 11) {
    System.out.println("Count is: " + count);
    count++;
}
```

FOR

```
for (initialization; termination;
    increment) {
    statement(s)
}
```

```
for (int i = 1; i < 11; i++) {
    System.out.println("Count is: " + i);
}
```

ENHANCED FOR

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
for (int number : numbers) {
    System.out.println("Number: " + number);
}
```

GOOD PRACTISE

Prefer for-each loops to traditional for loops

BRANCHING STATEMENTS

- `break`, `continue`, `return`

```
int[] randomNumbers = {-5, 12, 33, -64, 752, 9, -3, 112};
int searchFor = 9;
boolean found = false;
// break
for (int number : randomNumbers) {
  if (number == searchFor) {
    found = true;
    break;
  }
}
System.out.println("Found: " + found);

// continue
int positiveNumbers = 0;
for (int number : randomNumbers) {
```