2. Java Basics

1. Data Types

Java Data Types

- Primitive
  - Boolean
  - Numeric
    - Integer
    - Float-point
  - Char

- Reference
  - Array
  - Class
  - Interface

Boolean Type

- Type **boolean**
- Two possible values: **true, false**
- Use this data type for simple flags
- **Not compatible** with other types (integer!)
- Even explicit cast is impossible
- Its "size" isn't something that's precisely defined

Boolean Operators

- =	assignment
- == !=	equal to, not equal to
- !	NOT
- &&	AND
- ||	OR
- ?:	if-then-else
- &	bitwise AND
- |	bitwise OR

If-Then-Else Boolean Operator

- **expression1 ? expression2 : expression3**

- Examples:
  - BestReturn = Stocks > Bonds ? Stocks : Bonds;
  - LowSales = JuneSales < JulySales ? JuneSales : JulySales;
  - Distance = Site1 - Site2 > 0 ? Site1 - Site2 : Site2 - Site1;

AND Boolean Operator

```
boolean a = false;
boolean b = true;
boolean c = a && b;
boolean d = a & b;
```

Will we get the same results for c and d?

AND Boolean Operator

1. boolean a = false;
2. boolean b = true;
3. boolean c = a && b;

Operation && calculates first operand. If it equals false, then returns false without second operand calculation

4. boolean d = a & b;

Operation & calculates both operands and then returns the result

Integer Types

Type	Bytes	Min	Max
byte	1	-128	127
short	2	-32768	32767
int	4	-2 147 483 648	2 147 483 647
long	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

All integer type are singed integer types

int is approximately in interval -2E9 to 2E9

long is approximately in interval -9E18 to 9E18

Integer Literals

- Decimal constant should start with nonzero digit
- Leading zero means octal constant (*so 8 and 9 digits are impossible*)
- Leading 0x means hexadecimal constant (you can use A-F or a-f as digits)
- Long constant ends with L or l symbols.
- Any number of underscore characters (_) can appear anywhere between digits in a numerical constants (**since Java 7 only!**)

Integer Arithmetic Operations

- + add
- - subtract
- * multiply
- / divide
- % get reminder

Integer Addition

- byte a = 120;
- byte b = 10;
- byte c = (byte)(a + b);

What will be c value?

Why we use (byte)(a + b)?

Integer Arithmetic Operations

- If one operand has long type then other operand is converted to long. Otherwise both operands are converted to int type.
- The result of an operation has int type if it value does not need long type.

Integer Assignment

- The integer assignment performs implicit type conversion if neither accuracy nor value is loss (e.g. int = byte or long = int)
- If implicit cast is impossible then explicit cast is needed, otherwise compilation error will occur (e.g byte = (byte)int)

Java Overflow And Underflow

- In Java arithmetic operators don’t report overflow and underflow conditions
- When the result of an arithmetic integer operation is larger than 32 bits then the low 32 bits only taken into consideration and the high order bits are discarded
- The same with long type (64 bits)
- It’s a shame of Java

The Overflow Problem

- In Java arithmetic overflow will **never** throw an exception

```
long a = 9223372036854775806L;
long b = 2L;
long c = a + b;
c =  -9223372036854775808L
```

Integer Division

```
x = a / b
r = a % b
```

```
int a = 20;
int b = 3;
int c = a / b;
int d = a % b;
```

What will be c and d values?

Integer Division

Division by 0 leads to runtime ArithmeticException:

```
int a = 5;
int b = 0;
int c = a / b;
```

The Integer Unary Operators

- +   Unary plus operator
- -   Unary minus operator
- ++   Increment operator
- --   Decrement operator
- For pre-increment and pre-decrement (i.e., ++a or --a), the operation is performed and the value is produced.
- For post-increment and post-decrement (i.e., a++ or a--), the value is produced, then the operation is performed.

What will be a value?

- int x = 8;
- int a = x++ / x;

What will be done?

- int c = 10;
- int d = c+++++c;

What will be done?

```
int c = 10;
int d = c++ + ++c;
```

Bitwise Operators

- ~ inverts a bit
- & bitwise AND
- |bitwise OR
- ^ bitwise inclusive OR

Bitwise Operators

```
int a = 45;
int b = 34;
int c = a ^ b;
```

What will be c value?

```
int d = c ^ b;
```

What will be d value?

Bit Shift Operators

- << signed left shift operator
- >> signed right shift operator
- >>> right shift operator

Bit Shift Operators

```
int a = 45;
int b = a >> 3;
b = ?
int c = a << 3;
c = ?
```

Integer Assignment Operators

- =
- +=, -=, *=, /=
- <<=, >>=, >>>=
- &=, |=, ^=

Integer Assignment Operators

- x += 1; instead x = x + 1;
- a *= 5; instead a = a * 5;

The Equality and Relational Operators

• ==	equal to
• !=	not equal to
• >	greater than
• >=	greater than or equal to
• <	less than
• <=	less than or equal to

Float point Data Types

- float – 32 bit (± 1E38, 7-8 dec. precision)
- double – 64 bit (± 1E308, 16-17 dec. precision)

Accordingly IEEE 754-1985 standard

Float point Arithmetic Operations

+	add
-	subtract
*	multiply
/	divide

Float point Arithmetic Operations

- If one operand has double type then other operand is converted to double and result will be double type.
- If one operand has float type and other operand has any type differs from double then other operand is converted to float and result will be float type

What will be c and d value?

- double a = 2.2;
- double b = -1.4;
- a = a - 2.2;
- double c = b / a;
- double d = Math.sqrt(b);

Special Float Point Values

- -Infinity
- +Infinity
- NaN

In previous code c = -Infinity, d = NaN

Precision Problem I

- double a = 2.0;
- double b = a - 1.1;

b will be 0.899999999999999999, not 0.9!

Precision Problem II

How many repetitions will be?

```
double d = 0.1;
while (d != 1.0) {
    System.out.println(d);
    d += 0.1;
}
```

Debugging in Eclipse

- Start debugging: press Debug icon and use F6 key for stepped debugging
- Use Cntr + Shift + B for breakpoint creation
- Use Cntr + R to run application to the next breakpoint

Precision Problem Source

Above precision problems caused by the fact that finite decimal fraction 0.1 is infinite periodical binary fraction:

$$\frac{1}{10} = \frac{1}{16} + \frac{1}{32} + \frac{1}{16}\left(\frac{1}{10}\right)$$

So 0.1 can be represented as binary fraction in a computer only approximately.

Float point Literals

Here are possible formats for float point constants

- 1003.45
- .00100345e6
- 100.345E+1
- 100345e-2
- 1.00345e3
- 0.00100345e+6

Suffix f(F) means float constant, suffix d(D) – double constant. Constant without suffix - double

The Float point Unary Operators

- + Unary plus operator
- - Unary minus operator
- ++ Increment operator
- -- Decrement operator

Float point Assignment Operators

- **=**
- **+=, -=, *=, /=**

The Equality and Relational Operators

- **==** equal to
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

Char Type

- The char data type is a single 16-bit Unicode character
- Char data can be processed as unsigned short integers (0 – 65535) too.

Char Literals

- A symbol: 'a', 'A', '9', '+', '_', '~' (except \)
- Unicode symbol: '\u0108'
- Escape sequences '\b' '\t' '\n' '\f' '\r' '\"' '\'' '\\'

Don’t confuse char and string literals (e.g. ‘r’ and “r”)!

The \uxxxx notation can be used anywhere in the source to represent unicode characters

Char Examples

```
char c = 'g';
System.out.println(++c);

char r = (char)(c ^ 32);
```

Expressions. Operator precedence

. [] () + - ~ ! ++ -- instanceof * / % + - << >> >>> < <= >= > == !=	& ^ | && || ?: = op=

Casting (1 of 2)

- Any integer type can be casted to any other primitive type except boolean
- Casting from larger integer type to smaller (from long to short for example) can lead to data loss
- Casting from integer type to float point type can lead to precision loss (if integer is not power of 2)

Casting (2 of 2)

- Char type casting is the same as short integer type casting.
- Casting from float or double types to integer types returns integer part of the value without rounding

Casting operators (1 of 2)

- Implicit casting:

```
byte b = 18;
int a = b;
```

- Explicit casting:

```
int a = 18;
byte b = (byte)a;
```

Casting operators (2 of 2)

- int b = 168;
  double a = b;
- float p = 18.94f;
  byte b = (byte)p;   // b = 18

Manuals

- Learning the Java Language. Language Basics
- Thinking in Java. Operators.