Primitive data types

- All primitive values belong to one of eight primitive types

  `byte` `short` `int` `long` `float`
  `double` `char` `boolean`

- Primitive data types use a fixed number of bytes
  - four of these types designate different sizes of bounded integers: `byte`, `short`, `int`, `long`
- A programmer can not create new primitive data types
- Any data type you invent will be a type of *object*
- Most commonly used types in practice: `int`, `boolean`, and `double`

Java primitive data types

Primitive Type	Description	Range
`byte`	8-bit integer	-128 to 127
`short`	16-bit integer	-32768 to 32767
`int`	32-bit integer	-2147483648 to 2147483647
`long`	64-bit integer	-2^{63} to $2^{63}-1$
`float`	32-bit floating point	10^{-46} to 10^{38}
`double`	64-bit floating point	10^{-324} to 10^{308}
`char`	Unicode character	
`boolean`	Boolean variable	`false` and `true`

More on Data Types

- Trade-off b/w memory used and what size value the data type can store
- Single bit: 2 values, 2 bits: 4 values, 3 bits: 8 values, and so on. N bits: 2^N values

  - byte uses 8 bits => 2^8= 256 values (-128 to 127)
- Signed: both +ve and -ve values
- Integers: values stored in binary notation
- Floating point numbers: bits divided to store sign, mantissa, and exponent

  Example: 2.99792458×10^8

Variable Declaration

Have to declare all variables before using them!

```
int number;
```

1) new variable of type “int”
2) having the name “number”

Examples

- `int x, y, z;`
- `int sum = 0;`
- `float f;`
- `double pi = 3.14;`
- ```
  char first  = 'T',
       middle = 'L',
       last   = 'B';
  ```
- ```
  char first  = 'T';
  char middle = 'L';
  char last   = 'B';
  ```

What's wrong in these ?

1) Int x;
2) float y
3) int float;
4) int 2good;
5) int yes&no;

Arithmetic Expressions

- Expressions: collections of operands (constants and variables) and operators
- Very similar to what you've seen in Math classes

Basic operators

Operator	Java	Description
Assignment	=	assigns rhs to lhs
Arithmetic	+, -, *, /, %	addition, subtraction, multiplication, division, remainder
Unary	-, ++, --	negative, auto increment, auto decrement
Equality	==, !=	equals to, not equals to
Relational	<, <=, >, >=	less than, less than or equals to, greater than, greater than or equals to
Logical	&&, ||, !	AND, OR, NOT

Examples

```
int answer = 10 - 4;
```

Division is different, depending on integer/floating point

- If both are integers (byte, short, int, long)=> integer division

  Example: int answer = 5/2; (remainders/fractions are dropped:answer will be 2)

- If one or both are floating point => floating point division

  Example: double answer = 5/2.0; (fraction parts saved: answer will be 2.5)

Remainder operator (mod operation): returns remainder

  Example: int answer = 10%3; (answer will be 1)

More Examples

1) X=2;

   X++; (means X=X+1 –> so X will be 3)

2) a==b (checks if a is equal to b)

3) a!=b (checks if a not equal to b)

4) (a==b) &&(c==d) (checks if a = b and if c=d)

   (what if a=2, b=2, c=3, d=4 ?)

5) (a==b) || (c==d) (checks if a = b or if c=d)

   (what if a=2, b=2, c=3, d=4 ?)

6) if(!a) (checks if a==0)

Operator precedence

- **Evaluate** `a + b * c`
  - **multiplication first?** `a + (b * c)`
  - **addition first?** `(a + b) * c`
- **Java solves this problem by assigning priorities to operators** (operator precedence)
  - **operators with high priority are evaluated before operators with low priority**
  - **operators with equal priority are evaluated left to right**

Operator priority (highest to lowest)

1. ( )
2. * / %
3. + -
4. =

When in doubt, use parentheses

- `a + b * c = a + (b * c)`
  - because * has higher priority than +
- To perform the + operation first we need to use parentheses
  - `(a + b) * c`
- If in any doubt use extra parentheses to ensure the correct order of evaluation
  - parentheses are free!
  - cause no extra work for the computer
  - only make it easier for you to work out what is happening

Examples

- Java adheres to traditional order of operations
- * and / have higher priority than + and –

```
int x = 3 + 5 * 6;          (x = 33)
int y = (3 + 5) * 6;        (y = 48)
```

- Parentheses are free, use them liberally

```
int z = ((3 + 5) * (6));    (z = 48)
```

- Equal priority operations are evaluated left-to-right in the absence of parentheses

```
int w = 3 * 4 / 2 * 6;            (w = 36)
int x = 3 * 4 / (2 * 6);    (x = 1)
int y = 3 * 4 + 2 * 6;            (y = 24)
int z = 3 * (4 + 2) * 6;    (z = 108)
```

Syntax and semantics

- **Addition, subtraction:** `+` **and** `-`, `int` **and** `double`

```
int x = 21+4;          (x = 25)
double y = 14.1-2;     (y = 12.1)
```

- **Multiplication:** `*`, `int` **and** `double`

```
int x = 21*4;          (x = 84)
double y = 14.1*2.5;   (y = 35.25)
```

- **Division:** `/`, **different for** `int` **and** `double`

```
int x = 21/4;          (x = 5)
double y = 21/4;       (y = 5.0)
double y = 21/4.0;     (y = 5.25)
```

- **Modulus:** `%`, **only for** `int`

```
int x = 21%4;          (x = 1)
```