Java’s Number Types

Number Types

- `int:` integers, no fractional part

```
1, -4, 0
```

- `double:` floating-point numbers (double precision)

```
0.5, -3.11111, 4.3E24, 1E-14
```

Number Types

- A numeric computation overflows if the result falls outside the range for the number type

```
int n = 1000000;
System.out.println(n * n); // prints -727379968
```

- Java: 8 primitive types, including four integer types and two floating point types

Primitive Types

Type	Description	Size
`int`	The integer type, with range –2,147,483,648 . . . 2,147,483,647	4 bytes
`byte`	The type describing a single byte, with range –128 . . . 127	1 byte
`short`	The short integer type, with range –32768 . . . 32767	2 bytes
`long`	The long integer type, with range –9,223,372,036,854,775,808 . . . –9,223,372,036,854,775,807	8 bytes

Primitive Types

Type	Description	Size
`double`	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
`float`	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
`char`	The character type, representing code units in the Unicode encoding scheme	2 bytes
`boolean`	The type with the two truth values `false` and `true`	1 byte

Number Types: Floating-point Types

- Rounding errors occur when an exact conversion between numbers is not possible

```
double f = 4.35;
System.out.println(100 * f); // prints 434.99999999999994
```

- Java: Illegal to assign a floating-point expression to an integer variable

```
double balance = 13.75;
int dollars = balance; // Error
```

Number Types: Floating-point Types

- `Casts`: used to convert a value to a different type

```
int dollars = (int) balance; // OK
```

Cast discards fractional part.

- `Math.round` converts a floating-point number to nearest integer

```
long rounded = Math.round(balance); // if balance is 13.75, then
                                    // rounded is set to 14
```

Syntax 4.1: Cast

```
(typeName) expression
```

Example:

```
(int) (balance * 100)
```

Purpose:

To convert an expression to a different type

Constants: `final`

- A `final` variable is a constant
- Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: use all-uppercase names for constants

```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;
payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
   + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
```

Constants: `static final`

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as `static` and `final`
- Give `static final` constants public access to enable other classes to use them

```
public class Math
{
   . . .
   public static final double E = 2.7182818284590452354;
   public static final double PI = 3.14159265358979323846;
}


double circumference = Math.PI * diameter;
```

Syntax 4.2: Constant Definition

In a method:

```
final typeName variableName = expression ;
```

In a class:

```
accessSpecifier static final typeName variableName = expression;
```

Example:

```
final double NICKEL_VALUE = 0.05;
public static final double LITERS_PER_GALLON = 3.785;
```

Purpose:

To define a constant in a method or a class