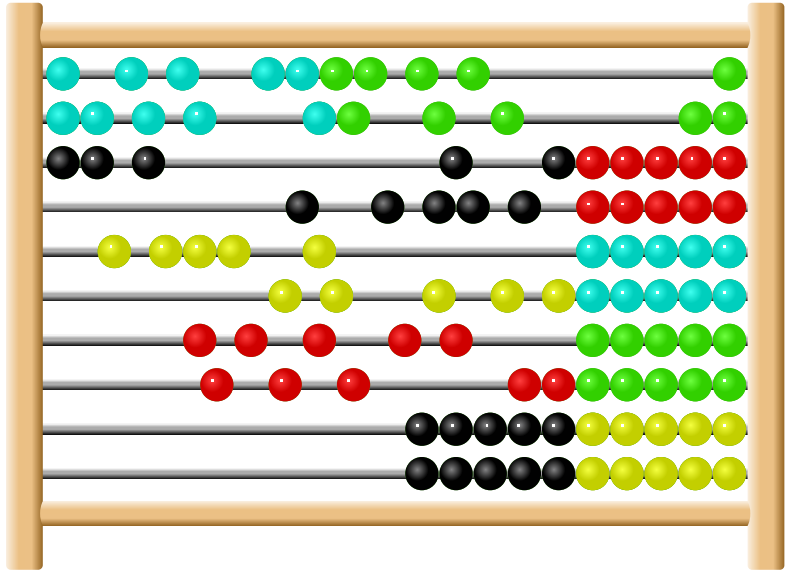
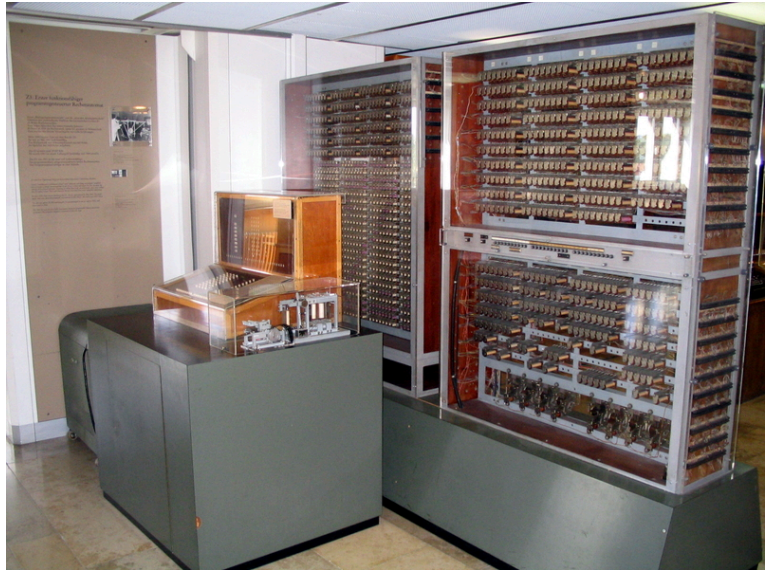
Manual calculation: Abacus

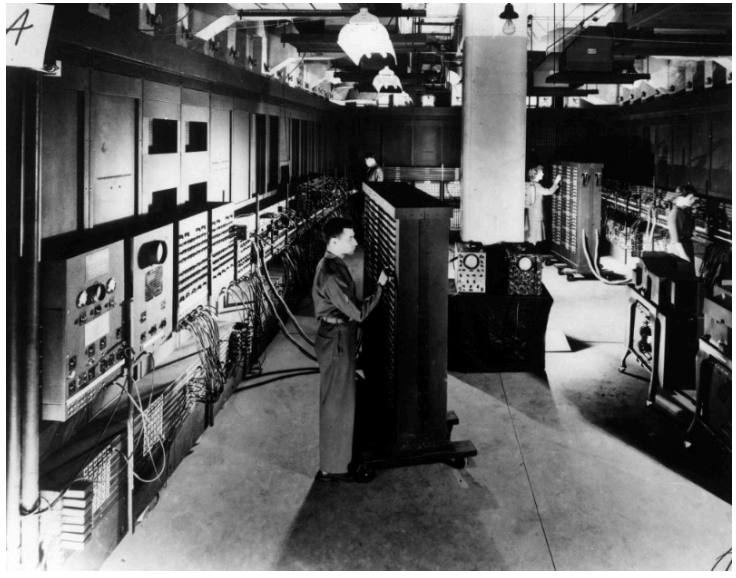


Mechanical calculation: Cash register

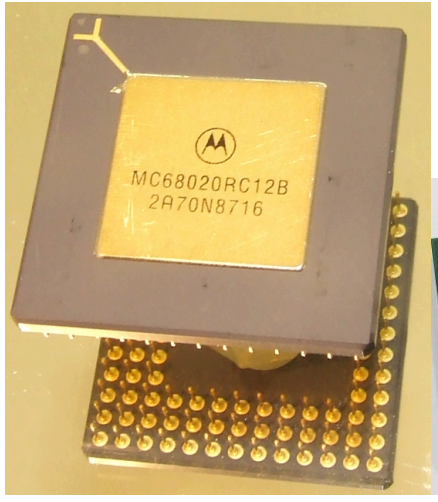
Electromechanical calculation: Zuse Z3

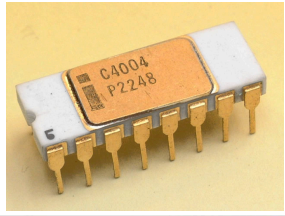


Vacuum Tube: Eniac

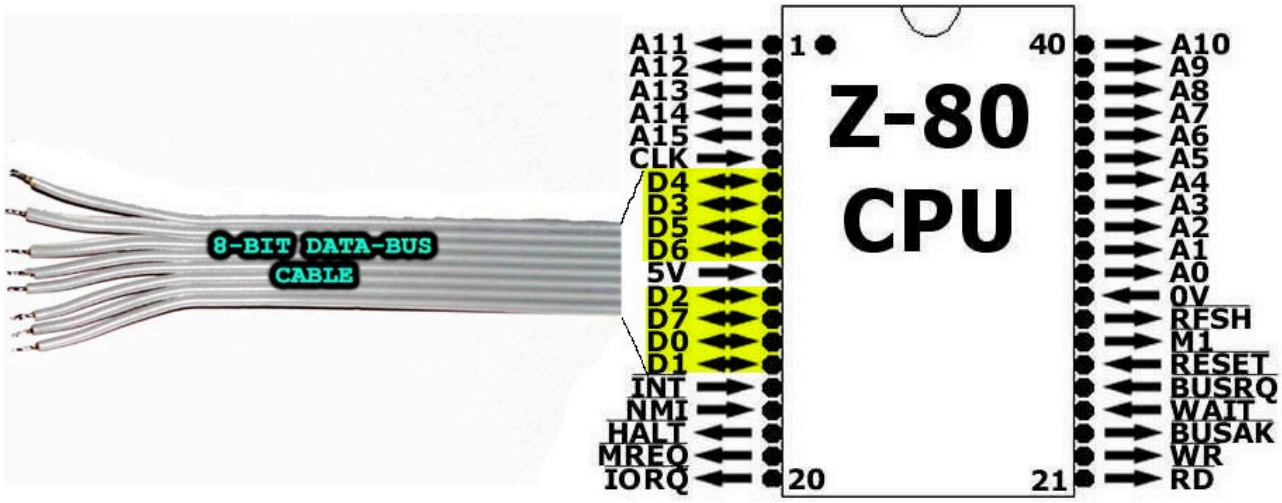


Transistor: Microprocessor ICs





Z80 8-bit data bus



Progress in hardware 1

Processor	Year	Address/ data bus	Transistors	Clock rate
Intel 4004	1971	12 / 4	2,300	740 kHz
Zilog Z80	1976	16 / 8	8,500	2.5 MHz
Motorola 68020	1984	32 / 32	190,000	12.5 MHz

Progress in hardware 2

Processor	Year	Address/ data bus	Transistors	Clock rate
Six-core Opteron	2009	64 / 64	904,000,000	1.8 GHz
Core i7 Broadwell	2016	64 / 64	3,200,000,000	3.6 GHz
Apple's ARM M1 Ultra	2022	64 / 64	114,000,000,000	3.2 GHz

Simple facts:

There are only **10** types of people in the world:

Those who understand binary and those who don't.

Unsigned 3 bit integer representation

0

0 0 0

$0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2\#$

Unsigned 3 bit integer representation

0	0	0	0	$0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2\#$
1	0	0	1	$0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2\#$

Unsigned 3 bit integer representation

0	0	0	0	$0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$
1	0	0	1	$0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
2	0	1	0	$0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

Unsigned 3 bit integer representation

0	0	0	0	$0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2\#$
1	0	0	1	$0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2\#$
2	0	1	0	$0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2\#$
3	0	1	1	$0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2\#$

Unsigned 3 bit integer representation

0	0	0	0	$0\cdot2^2 + 0\cdot2^1 + 0\cdot2^0$
1	0	0	1	$0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$
2	0	1	0	$0\cdot2^2 + 1\cdot2^1 + 0\cdot2^0$
3	0	1	1	$0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$
4	1	0	0	$1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0$
5	1	0	1	$1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$
6	1	1	0	$1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0$
7	1	1	1	$1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$

Binary system addition

Within limits: o.K.		**Caution: Overflow!**
010 2 +011 +3 ---- --- 101 5		100 4 101 +5 ------ --- discarded (1)001 1 by 3 bit representation

3 bit two-complement representation

-4

3 bit two-complement representation

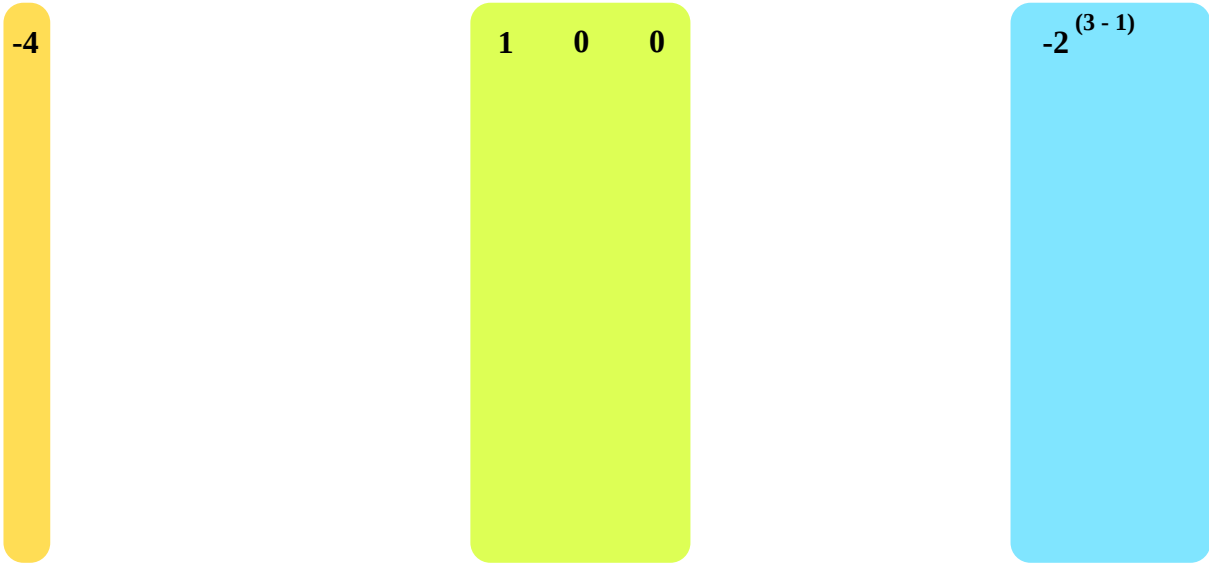
-4

Converting decimal -4 to 3- bit two-complement rule:

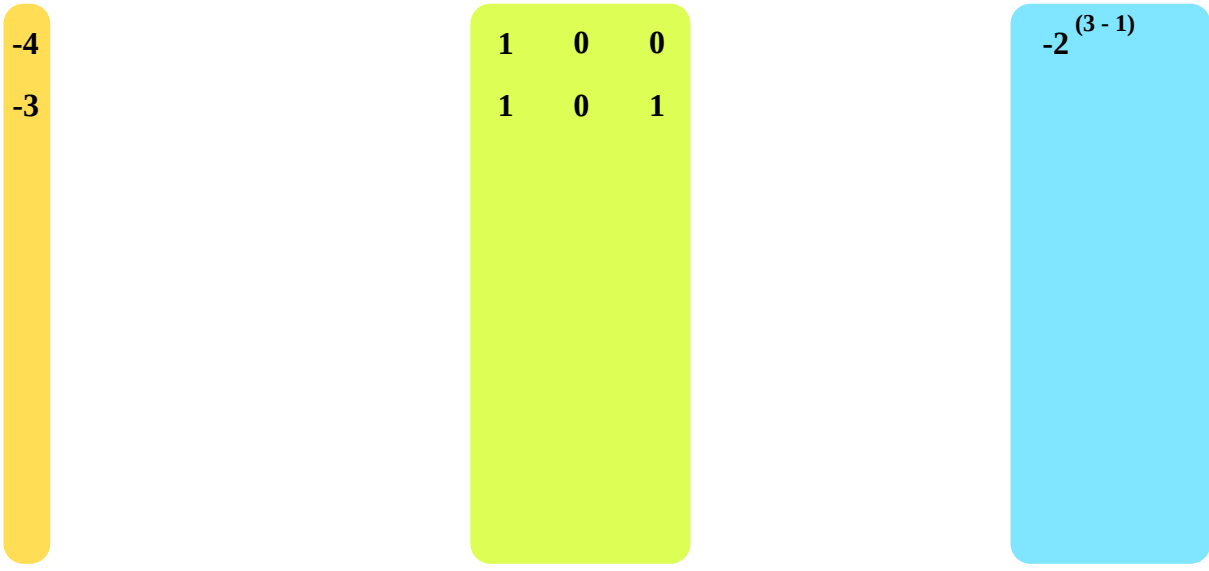
1. Start from positive value 4: --> 100
2. Invert all bits (1-complement) --> 011
3. Add value 1 (001): --> 100

Note: The number of bits matters:
In 8-bit two-complement -4 is being represented by 11111100

3 bit two-complement representation



3 bit two-complement representation



3 bit two-complement representation

-4	1	0	0
-3	1	0	1
-2	1	1	0
-1	1	1	1
0	0	0	0

$-2^{(3-1)}$

3 bit two-complement representation

-4	1	0	0
-3	1	0	1
-2	1	1	0
-1	1	1	1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1

$-2^{(3-1)}$

$2^{(3-1)} - 1$

3 bit two complement rationale: “Usual” addition

Within limits: o.K.		**Caution: Overflow!**
101 -3 +010 +2 ---- --- 111 -1		100 -4 101 -3 ---- --- 1001 1

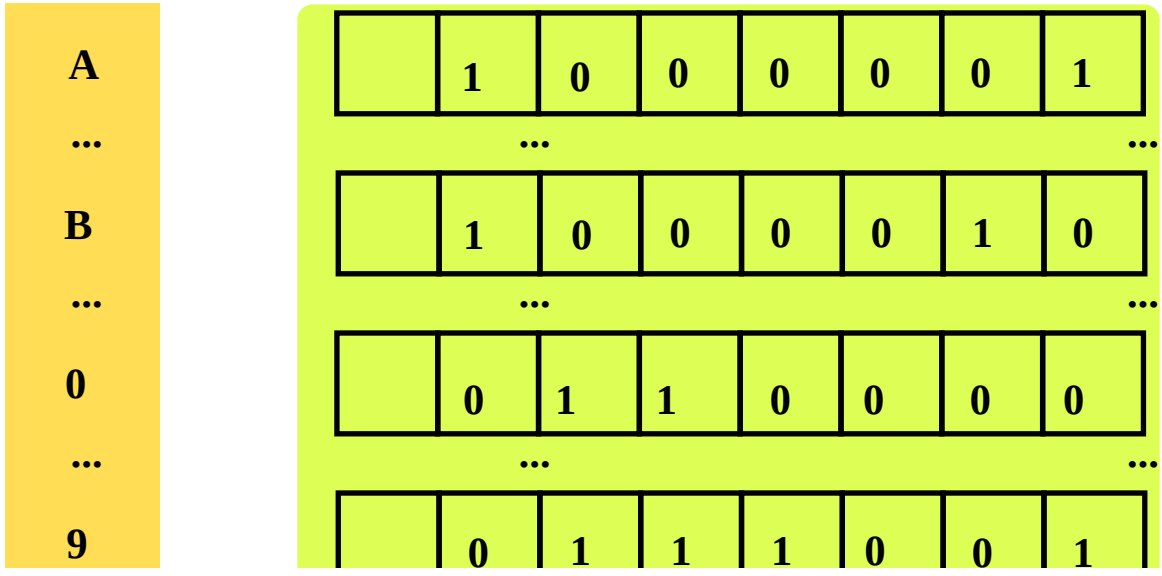
Signed 8 bit integer binary representation

-128	1	0	0	0	0	0	0	0	$-2^{(8-1)}$
-127	1	0	0	0	0	0	0	1	
-1	1	1	1	1	1	1	1	1	
...					...				
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	1	
...					...				
126	0	1	1	1	1	1	1	0	
127	0	1	1	1	1	1	1	1	$2^{(8-1)} - 1$

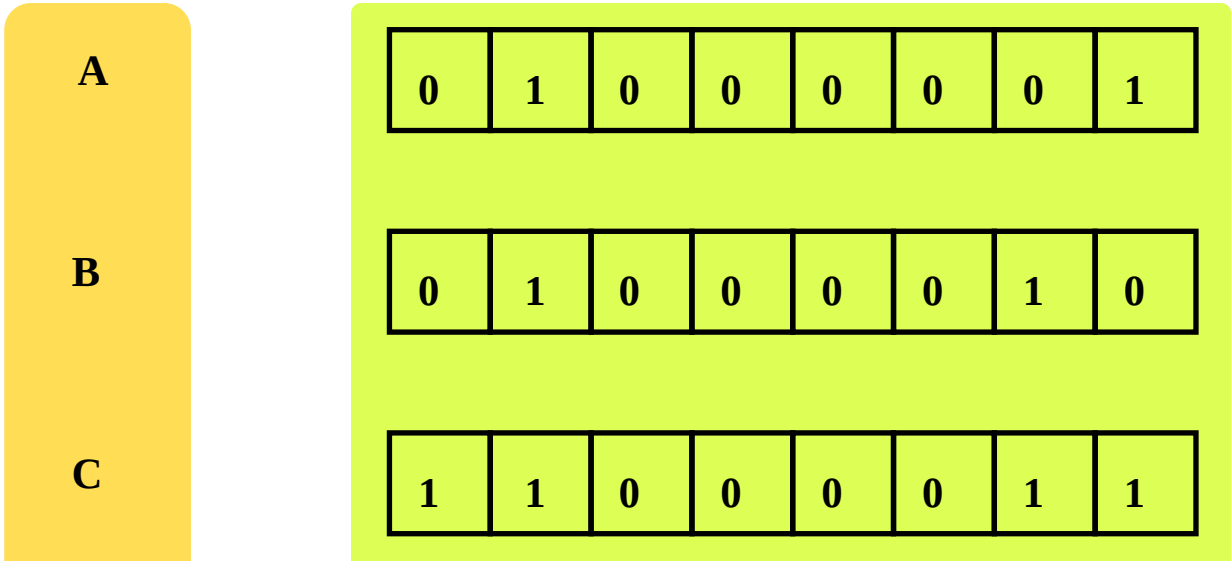
Related exercises

Exercise 5: Hotel key cards

7-bit ASCII



7-bit ASCII with even parity bit



Western European characters: ISO Latin 1 encoding

A	0	1	0	0	0	0	0	1
µ	1	0	1	1	0	1	0	1
¾	1	0	1	1	1	1	1	0
Å	1	1	0	0	0	1	0	1
Ç	1	1	0	0	0	1	1	1

Unicode UTF-8 samples

A


0 1 0 0 0 0 0 1
0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0

65
1692
128526

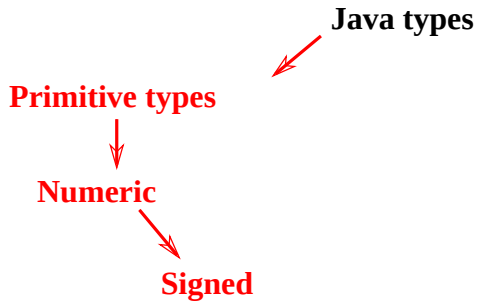
Java types

Java types

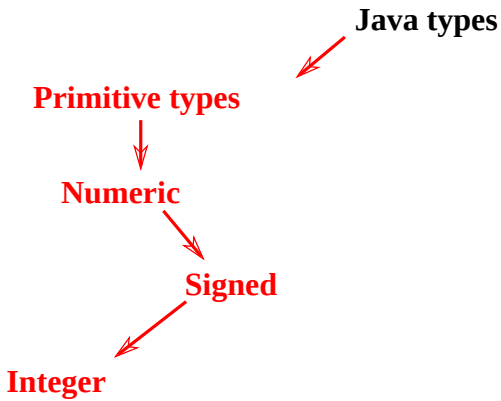
Java types

Java types

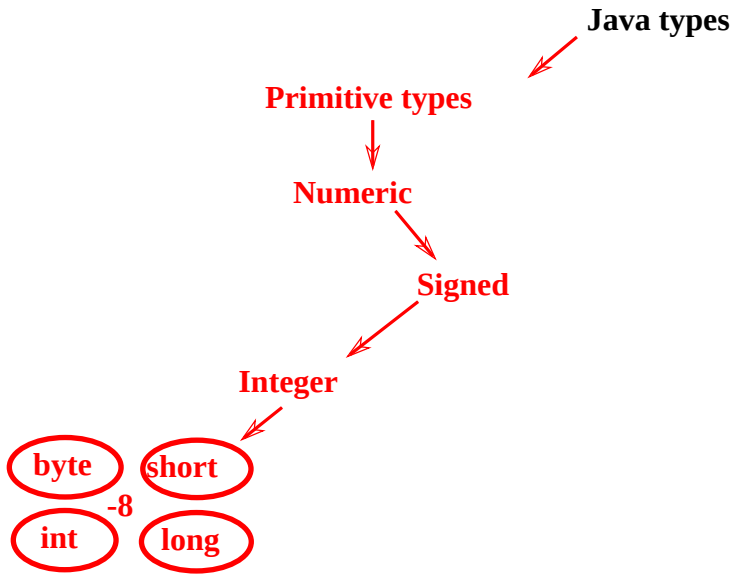
Java types



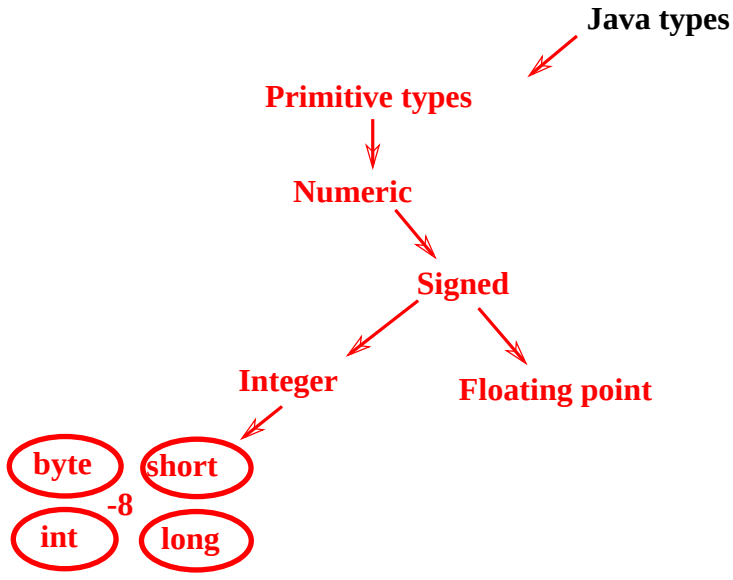
Java types



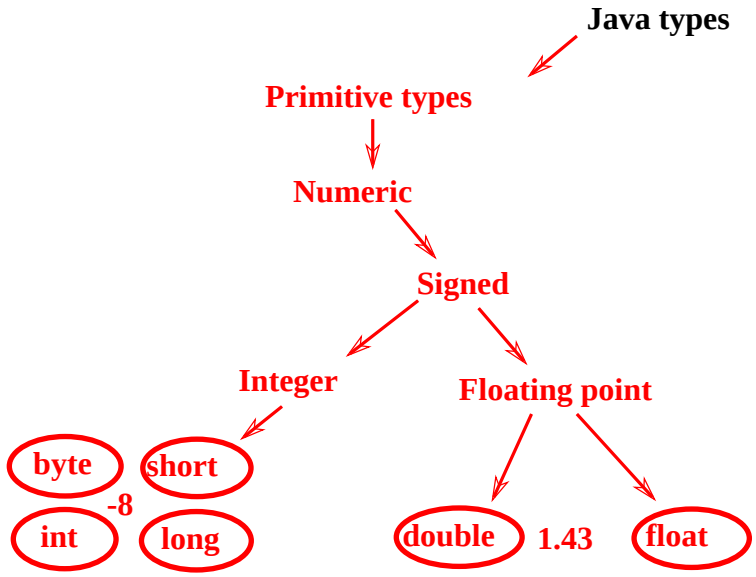
Java types


Java types
Primitive types
Numeric
Signed
Integer
byte
short
-8
int
long

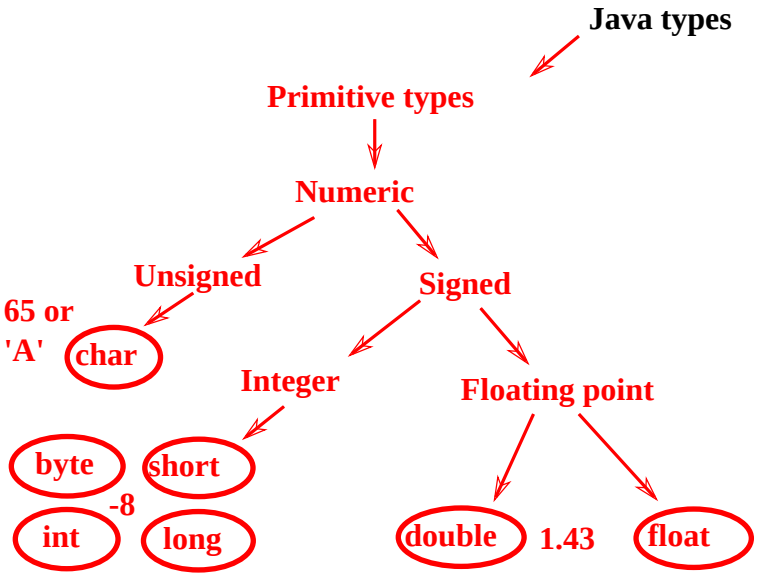
Java types


Java types
Primitive types
Numeric
Signed
Integer
Floating point
byte
short
-8
int
long

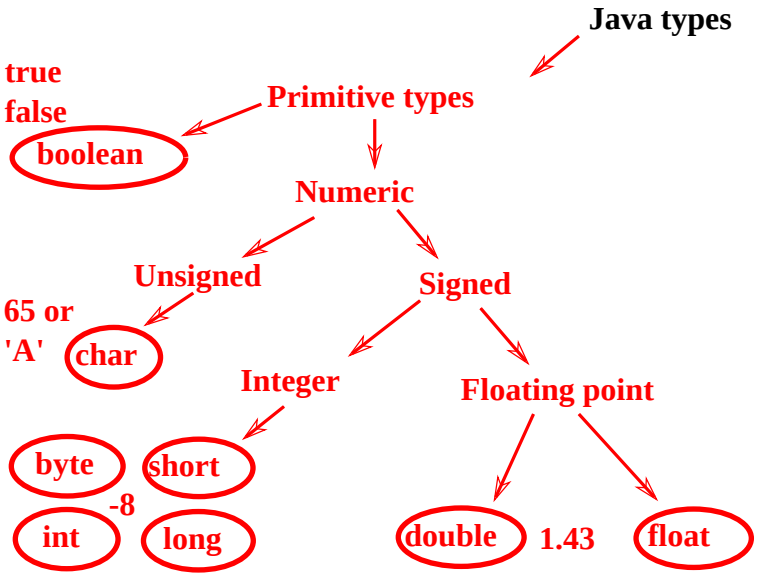
Java types


Java types
Primitive types
Numeric
Signed
Integer
Floating point
byte
short
-8
int
long
double
1.43
float

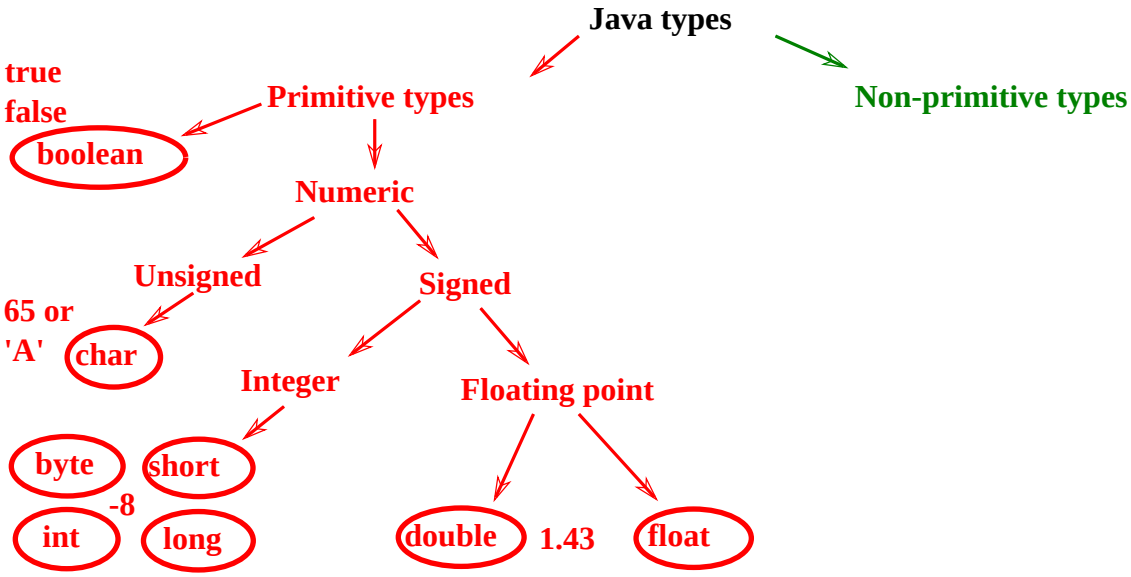
Java types


Java types
Primitive types
Numeric
Unsigned
Signed
65 or
'A'
char
Integer
Floating point
byte
short
-8
int
long
double
1.43
float

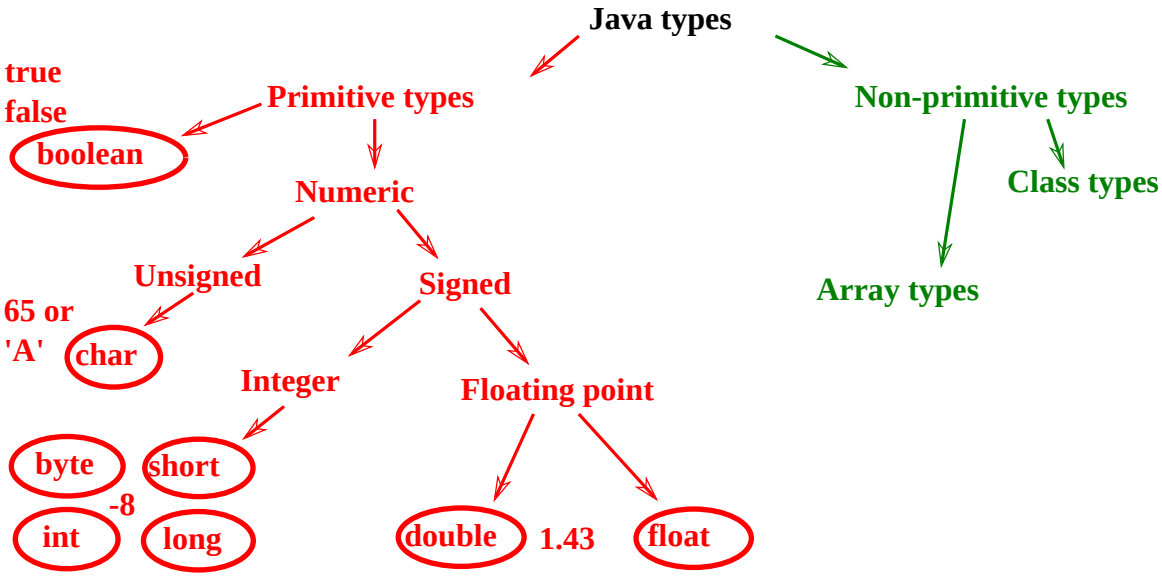
Java types


Java types
true
false
Primitive types
boolean
Numeric
Unsigned
Signed
65 or
'A'
char
Integer
Floating point
byte
short
-8
int
long
double
1.43
float

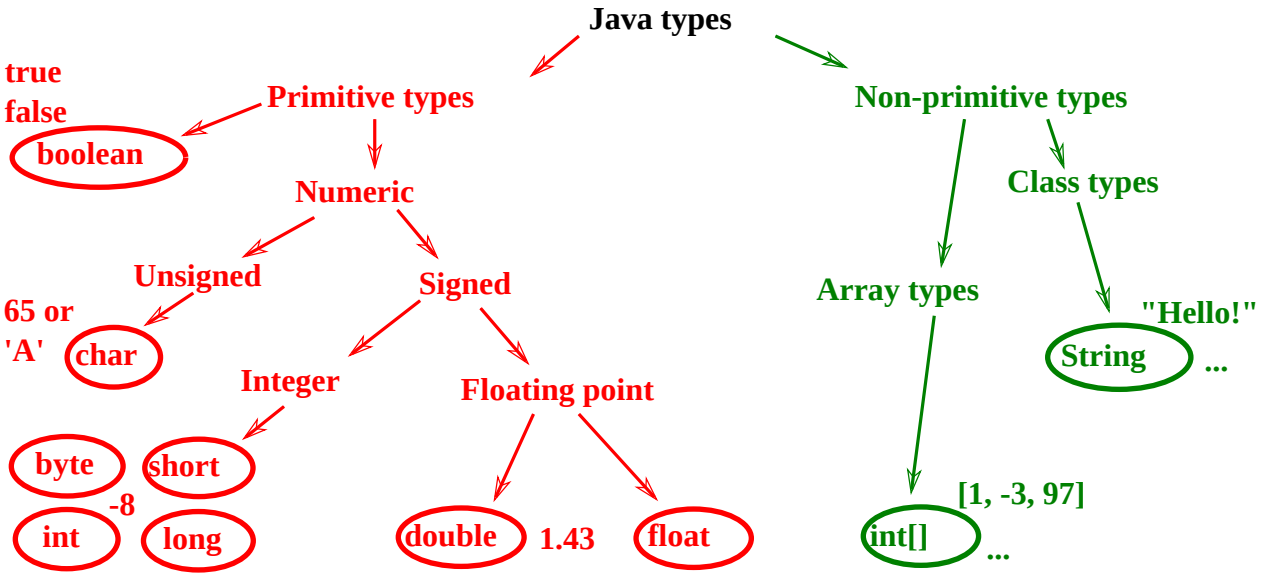
Java types


Java types
Primitive types
Non-primitive types
true
false
boolean
Numeric
Unsigned
Signed
65 or
'A'
char
Integer
Floating point
byte
short
-8
int
long
double
1.43
float

Java types


Java types
Primitive types
Non-primitive types
true
false
boolean
Numeric
Class types
Unsigned
Signed
Array types
65 or
'A'
char
Integer
Floating point
byte
short
-8
int
long
double
1.43
float

Java types


Java types
Primitive types
Non-primitive types
true
false
boolean
Numeric
Unsigned
Signed
65 or
'A'
char
Integer
Floating point
Array types
Class types
"Hello!"
String
...
byte
short
-8
int
long
double
1.43
float
[1, -3, 97]
int[]
...

Java primitive types, Part 1

Name	Bytes	Type	Range	Literal samples
`byte`	1	Signed integer		-
`short`	2	Signed integer		-
`int`	4	Signed integer		0, 1, +1, -42, 0b1101, 017, 0xC
`long`	8	Signed integer		0L, 1L, +1L, -42L, 0b1101L, 017L, 0xCL

Java primitive types, Part 2

Name	Bytes	Type	Range	Literal samples
`char`	2	Unsigned integer		'A', '*', '-', 'Ç', '#' ...
`float`	4	Floating point	to	3.14f, 0.022f, -3.4E-24f
`double`	8	Floating point	to	3.14, 0.022, -3.4E-24, 3.14d,...
`boolean`	?	Logical value	not applicable	`true`, `false`

Variables: Handles to memory

Code:

Memory

Variables: Handles to memory

Code:

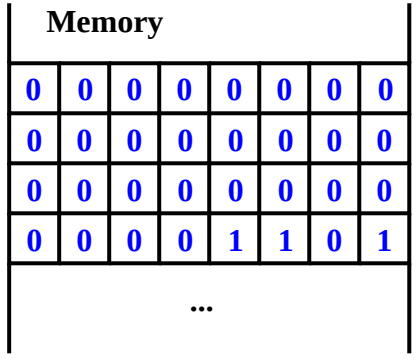
Memory

```
int a = 13;
```

Variables: Handles to memory

Code:

int a = 13;

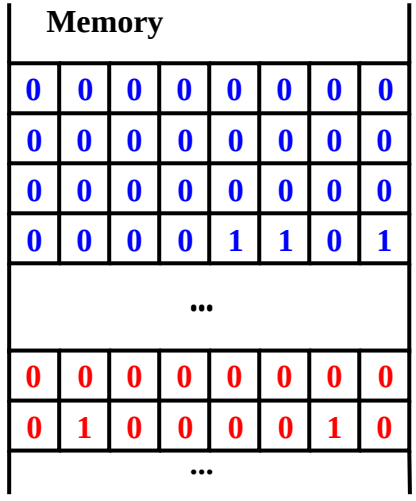


Variables: Handles to memory

Code:

```
int a = 13;

char c = 'B';
```



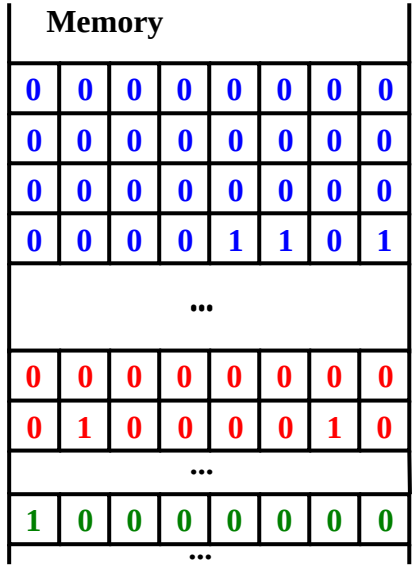
Variables: Handles to memory

Code:

```
int a = 13;

char c = 'B';

byte b = -128;
```



Variable declaration

```
// Variable declaration:
//     Variable's type is double
//     Variable's name is «pi» (identifier)

double pi;
{type name} {variable name} ;
```

Declare, assign and use

```
double pi;      // Variable declaration

pi = 3.1415926; // Assigning value to variable

// Print a circle's area of radius 2.0

System.out.println(pi * 2.0 * 2.0);
```

Combining declaration and initialization

Separate declaration and initialization

```
double pi;      // Declaration of variable pi

pi = 3.1415926; // Value assignment
```

Combined declaration and initialization

```
double pi = 3.1415926;
```

Multiple variables of same type

```
int a;
int b = 22;
int c;
```

being equivalent to either of:

Compact	Multiple lines

`int a, b = 22, c;`	`int a,`
	`    b = 22,`
	`    c;`

Identifier in Java™:

- Variable names.
- Class names
- Method names, e.g.:

```
public static void main(String [] args)
```

Identifier name examples:

Rules	Legal	Illegal

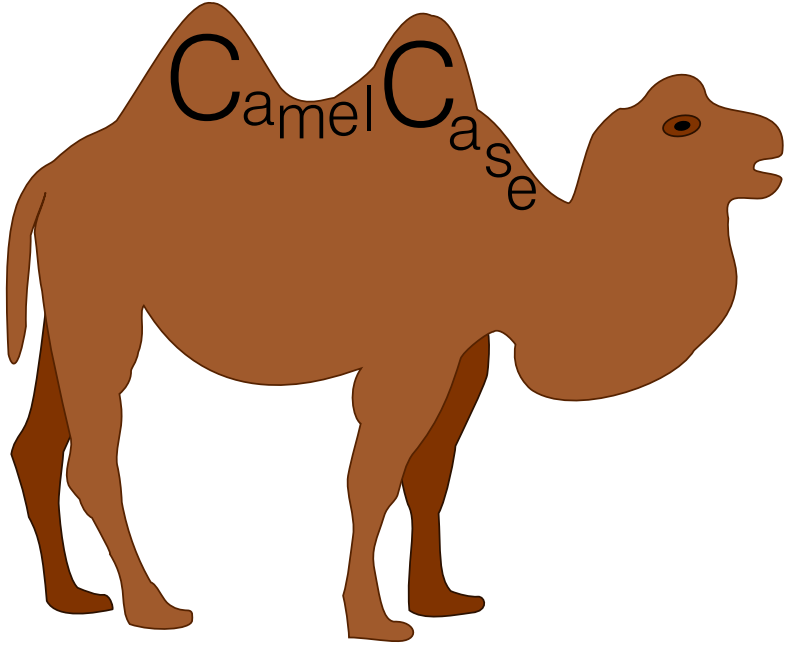
• Start with a letter, “_” or “$”, but not just a single “_”
• **May** be followed by letters, digits, “_” or “$”
• Must not match:
  • a Java™ keyword
  • “`boolean`” or “`null`” literal
	• `$test`
	• `_$`
	• `count`
	• `blue`
		• `2sad`
		• `_`
		• `switch`
		• `true`

Java™ keywords.

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

Variable naming conventions

- Start with a small letter like `africa` rather than `Africa`.
- Use “camel case” e.g. `myFirstCode`.



- Do not start with _ or $.

Constant variables

Modifier `final` prohibits changes:

```
final double PI = 3.1415926;
...
PI = 1.0; // Compile time error: Constant cannot be modified
```

Note

`final` variables by convention are being capitalized

Case sensitivity

Variable names are case sensitive:

```
int count = 32;
int Count = 44;
System.out.println(count + ":" + Count);
```

Resulting output:

```
32:44
```

Related exercises

Exercise 6: Legal variable names

Define before use

Correct:

```
double f;
f = -4.55;
```

Wrong:

```
f = -4.55;
double f;
```

Type safety

```
int i = 2;
int j = i;   // o.K.: Assigning int to int
long l = i;  // o.K.: Widening conversion

i = l;       // Wrong: Narrowing

boolean b = true;
i = b;       // Error: int and boolean are incompatible types
i = 4.3345   // Error: Cannot assign double to int
i = "Hello"; // Even worse: Assigning a String to an int
```

Compile time analysis

`byte b127 = 127; // o.K., static check`
`byte b128 = 128; // Wrong: Exceeding 127`
	Performing static range check

`int a = 120;`
 	
`byte b120 = a; // Error Incompatible types`
`// Required: byte`
`// Found:int`
	No static check on `int` variable's value

Related exercises

Exercise 7: Benefits of `final`

Forcing conversions

```
long l = 4345;

int i = (int) l; // Casting long to int
System.out.println("i carrying long: " + i);

double d = 44.2323;
i = (int) d; // Casting double to int
System.out.println("i carrying double: " + i);
```

```
i carrying long: 4345
i carrying double: 44
```

Watch out!

```
long l = 3000000000000000L;
int i = (int) l;
System.out.println("i carrying long:" + i);

double d = 443000000000000.0;
i = (int) d;
System.out.println("i carrying double:" + i);
```

```
i carrying long:-296517632
i carrying double:2147483647
```

Casting long to int

long a = 3000000000000000L;

Casting long to int

long a = 3000000000000000L;

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Casting long to int

long a = 3000000000000000L;

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Type cast long to int:

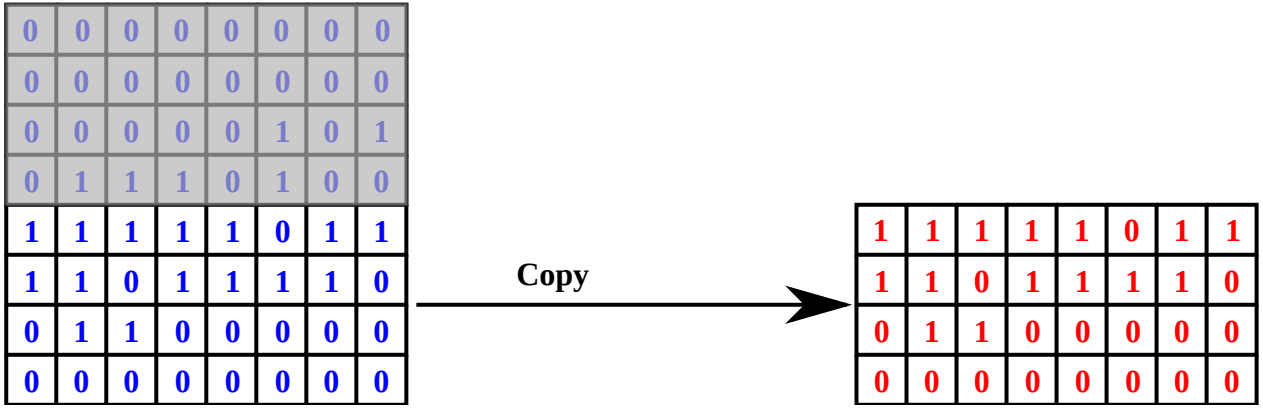
int b = (int) a;

Casting long to int

long a = 3000000000000000L;

Type cast long to int:

int b = (int) a;



Casting long to int

long a = 3000000000000000L;

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Type cast long to int:

int b = (int) a;

Value: -296517632

1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Don't worry, be happy ...

«C» programming language miracles:

```c
#include <stdio.h>

void main(void) {
  double measure = 65234.5435;
  short velocity;
  velocity = measure;
  printf("Velocity=%d\n", velocity);
}
```

Uups:

```
Velocity=-302
```

... and watch the outcome

Development costs:	~$7 billion.
Rocket and cargo	~$500 million.
	related video and explanations

From the report

The cause of the failure was a software error in the inertial reference system.

Specifically, a **64 bit floating point number** relating to the horizontal velocity of the rocket with respect to the platform was converted to a **16 bit signed integer**.

The number was larger than 32,767, the largest integer possible in a 16 bit signed integer, **and thus the conversion failed**.

Related video explanation

Related exercises

Exercise 8: «C» vs. Java.
Exercise 9: Assignment and type safety

Maximum and minimum values

Type	Bytes	Min value	Max value
byte	1		
char	2		
short	2		
int	4		
long	8		

Related exercises

Exercise 10: Inventing `tinyint`.
Exercise 11: An `int`'s minimum and maximum value

Dynamic typing in PERL

```
$test = 44;  # Assigning an int
print $test, "\n";

$test = "Jim"; # Assigning a string
print $test, "\n";

$cmp = 43.55; # A float

if ($test == $cmp) { # comparing string against float
   print "Equal\n";
} else {
   print "Different\n";
}
```

```
44
Jim
Different
```

Dynamic typing in PERL, part 2

```
$a = 2; # An integer
$b = 3; # Another integer

print '$a + $b = ', $a + $b, "\n";

$jim = "Jim";          # A string
print '$jim + $a = ' , $jim + $a, "\n";
```

```
$a + $b = 5
$jim + $a = 2
```

Using final

```
//Bad!
double pi = 3.141592653589793;
...
pi = -4; // Woops, accidential and erroneous redefinition

//Good
final double PI = 3.141592653589793;
...
PI = -4; // Compile time error:
         // Cannot assign a value to final variable 'pi'
```

Two categories of variables

Primitive type

```
int a = -15;
```

Possible types: All eight primitive Java™ types.

Reference type (based on classes)

```
GpsPosition start = new GpsPosition(48.7758, 9.1829);
```

Possible types: Arbitrary built-in or user defined classes.

Reference type examples

```
GpsPosition start = new GpsPosition(48.7758, 9.1829);
String name = "Simon";
LocalDate birthday = LocalDate.of(1990, Month.JULY, 5);
```

`float` and `double`

```
double d = 0.1;
```

float and double

double d = 0.1;

0	0	1	1	1	1	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0

float and double

double d = 0.1;

0	0	1	1	1	1	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0

float f = 0.1F

float and double

double d = 0.1;

0	0	1	1	1	1	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0

float f = 0.1F

0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1

Four ways representing 35

Code	Result

`System.out.println("Decimal "+ 35);`
`System.out.println("Binary " + 0b10_0011);`
`System.out.println("Hex "    + 0x23);`
`System.out.println("Octal "  + 043);`
	`Decimal 35`
	`Binary  35`
	`Hex 35`
	`Octal 35`

Choose your output representation

```
System.out.println("35 as Binary (Base 2):       " + Integer.toString(35, 2));
System.out.println("35 as Ternary (Base 3):      " + Integer.toString(35, 3));
System.out.println("35 as Octal (Base 8):        " + Integer.toString(35, 8));
System.out.println("35 as Hexadecimal (Base 16): " + Integer.toString(35, 16));
```

results in:

```
35 as Binary (Base 2):       100011
35 as Ternary (Base 3):      1022
35 as Octal (Base 8):        43
35 as Hexadecimal (Base 16): 23
```

Related exercises

Exercise 12: Pretty may not be pretty
Exercise 13: Strange output

Know your limits!

```
System.out.println(1000000000);    // o.K.
System.out.println(2147483647);    // o.K.: Largest int value 2^31 - 1

System.out.println(10000000000L);  // o.K.: Using type long
System.out.println(10000000000 );  // Compile time error: Integer number
                                   // larger than 2147483647 or
                                   // 2^31 - 1, Integer.MAX_VALUE)
```

Literal examples

```
System.out.println("Hello");     // A String literal

System.out.println(33452);       // An int literal

System.out.println(34.0223);     // A double (floating point) literal

System.out.println(2147483648L); // A long literal
```

`int` literals

```
System.out.println("Value 1: " + 29);
System.out.println("Value 2: " + 0b11101);
System.out.println("Value 3: " + 0x1D);
System.out.println("Value 4: " + 035);
```

```
Value 1: 29
Value 2: 29
Value 3: 29
Value 4: 29
```

`int` literals explained

Literal	Discriminator	Type	Value
29	base **10**	Decimal	
0b11101	**0b**, base **2**	Binary	
0x1D	**0x**, base **16**	Hexadecimal	
035	**0**, base **8**	Octal	

Java™ primitive literals

byte, short	-
char	'A', '\u0041'
int	29, 0b1_1101, 0x1D, 035, -29,
long	35L, 0b10_0011L, 0x23L, 043L, -35L,...
float	55.43F, 1.7E-23F, -17.F, 100_342.334_113F
double	55.43, 1.7 E -23, -17.
boolean	true, false

Java™ `String` and `null` literals

`String`	`"Hello"`, `"Greek Δ"`
	`"Greek \u0394"`
Arbitrary classes	`null`

Related exercises

Exercise 14: Poor mans ASCII table
Exercise 15: Integer value hexadecimal representation
Exercise 16: Binary literals
Exercise 17: Testing the limits (Difficult)
Exercise 18: Why using braces in `System.out.println(...)`?
Exercise 19: Composing strings of literals and variables
Exercise 20: Escaping double quotes
Exercise 21: Supplementary string exercises

Just kidding ...

```
int year = MMXIV; // Roman numerals representation

System.out.println("Olympic winter games: " + year);
```

Could this happen?

```
Olympic winter games: 2014
```

Strange things I

```
byte count = 91;   // o.K.

int i = 91;

byte count2 = i;   // Compile error: Incompatible types
                   // Required: byte Found: int

byte points = 130; // Compile error: Incompatible types
                   // Required: byte Found: int
```

Strange things II

```
final int i = 91;

byte count = i;   //  o.K.
```

Arithmetic overflow pitfalls

```
int count  = 2147483647;
int points = 2147483647;

int sum = count + points; ❶
System.out.println("Sum = "   +  sum);
```

Result:

Sum = -2	01111111_11111111_11111111_11111111 + 01111111_11111111_11111111_11111111 _____ 11111111_11111111_11111111_11111110

Limited precision

```
float float2Power31 = Integer.MAX_VALUE + 1f; // 2^31

float floatDoubleMAX_VALUE = 2 * float2Power31 * float2Power31 - 1f; // 2^63 - 1

System.out.format( "   Float value: %f\n", floatDoubleMAX_VALUE);
System.out.println("Expected value: "   +  Long.MAX_VALUE);
```

Result:

```
   Float value: 9223372036854776000.000000
Expected value: 9223372036854775807
    Difference:                  193
```

Nearest `float` to 2.1

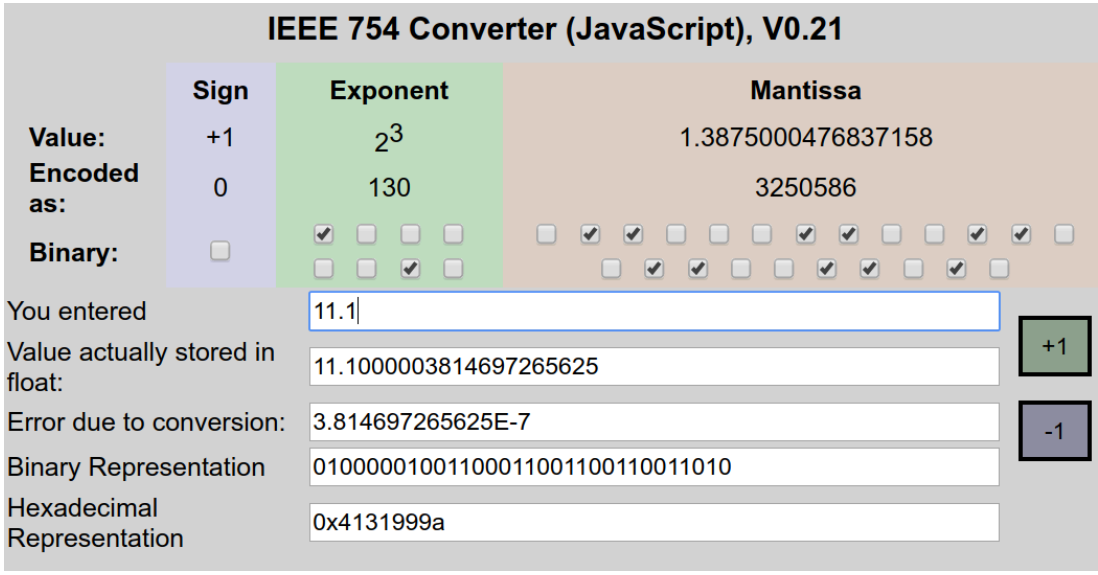
```
DecimalFormat df = new DecimalFormat("#.###############"); //Print 15 floating point digits

System.out.println(df.format(Float.intBitsToFloat(0b0_10000000_00001100110011001100110 )));

System.out.println(df.format(Float.intBitsToFloat(0b0_10000000_00001100110011001100111 )));
```

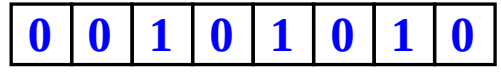
```
2.099999904632568
2.100000143051147
```

FloatConverter



Widening from `byte` literal to `short`

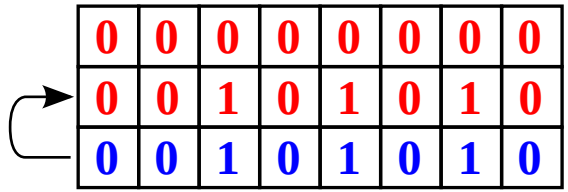
byte b = 42;



Widening from `byte` literal to `short`

byte b = 42;

short s = b;



Widening from `byte` literal to `short`

```
byte b = 42;

short s = b;

System.out.println(s);
```

0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0

Result: 42

Narrowing from `int` literal to `char` variable

System.out.println(65);

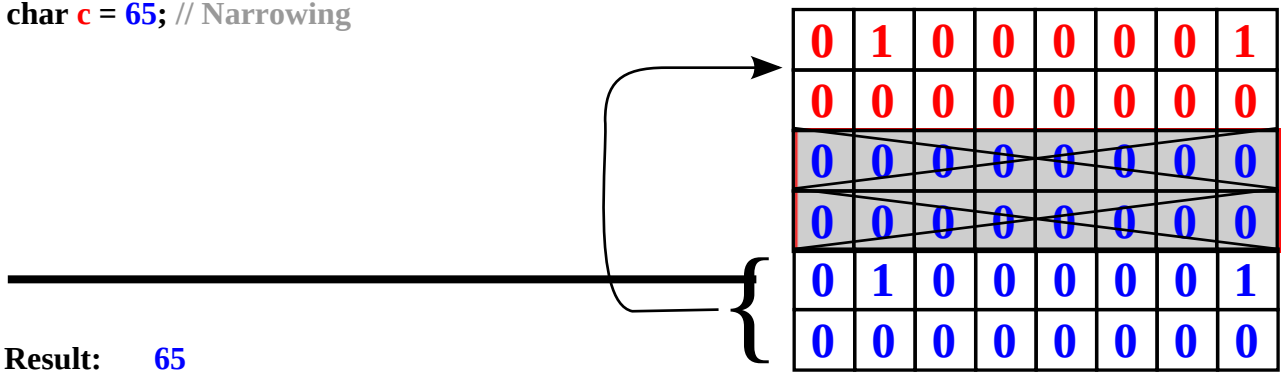
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Result: 65

Narrowing from `int` literal to `char` variable

```
System.out.println(65);

char c = 65; // Narrowing
```



Result: 65

Narrowing from `int` literal to `char` variable

```
System.out.println(65);

char c = 65; // Narrowing

System.out.println(c);
```

0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0

Result: 65
A

Narrowing from `int` literal to `char` variable

```
System.out.println(65);

char c = 65; // Narrowing

System.out.println(c);

//Type cast char to int (widening)
System.out.println((int) c);
```

Result:

```
65
A
65
```

0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0

A widening «ladder»

```
byte   b = 42;  // Narrowing: constant int literal to byte
short  s = b;   // Widening
int    i = s;   // Widening
long   l = i;   // Widening
float  f = l;   // Widening
double d = f;   // Widening
```

A narrowing «ladder»

```
double d = 14.23;
float  f  = (float) d;  // Narrowing
long   l  = (long)  f;  // Narrowing
int    i  = (int)   l;  // Narrowing
short  s  = (short) i;  // Narrowing
byte   b  = (byte)  s;  // Narrowing
```

Related exercises

Exercise 22: `int` and `char`
Exercise 23: `float` vs. `double`
Exercise 24: `int` to `char` narrowing problems
Exercise 25: Get a `byte` from 139
Exercise 26: Ariane, I miss you!
Exercise 27: Reducing `long` to `int` (difficult)

The binary plus operator

```
byte a = 4;

int result = a + 5;

System.out.println(result);
```

Operand
Type: byte

Operator

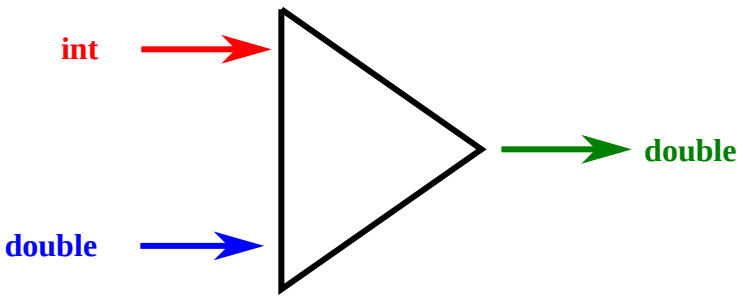
Operand
Type: int

Output: 9

Type: int

Binary operator output type

```
int a = 7;
double d = 2.3;
System.out.println(a + d);
```



Related exercises

Exercise 28: Calculating a circle's area
Exercise 29: Dividing values
Exercise 30: Strange things with operator ++
Exercise 31: Adding values
Exercise 32: Representational float and double miracles

Detecting arithmetic overflow (Java 8+)

```
try {
    int sum = Math.addExact(2147480000, 2147480000);
    System.out.println("sum = " + sum);
} catch (ArithmeticException ex) {
    System.err.println("Problem: " + ex.getMessage());
}

Problem: integer overflow
```

Dividing by zero

```
double f = 34.3 / 0;
System.out.println("Value: " + f);
```

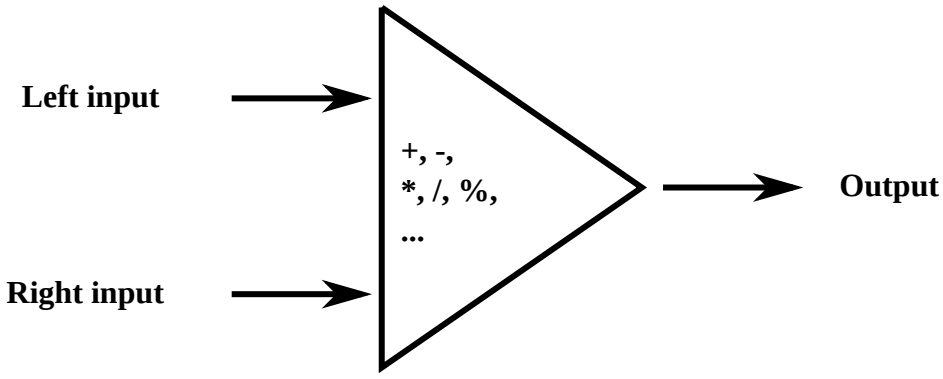
```
Value: Infinity
```

Watch out: “Silent” error!

Related exercises

Exercise 33: Expressions involving infinity

Generic binary operator



The modulus operator %

Get a division's remainder:

```
int nuggets = 11,
    diggers = 3;

System.out.println("Nuggets per digger:" + nuggets / diggers);
System.out.println("Remaining nuggets:" + nuggets % diggers);
```

```
Nuggets per digger:3
Remaining nuggets:2
```

Binary operator type examples

Left	Right	Out	Examples
`boolean`	`boolean`	`boolean`	`|, &, &&, ||, ^`
`int`	`int`	`int`	+, -, *, /, % (read here!)
`int`	`long`	`long`	
`byte`	`byte`	`int`	
`double`	`float`	`double`	
`int`	`float`	`float`	
`char`	`byte`	`int`	

No binary + operator yielding `byte`

```
byte a = 1, b = 2;

byte sum = a + b; // Error: Incompatible types. ❶
                  // Required: byte
                  // Found: int
```

Related exercises

Exercise 34: `int` to `short` assignment
Exercise 35: `int` to `short` assignment using `final`
Exercise 36: Calculating a circle's area avoiding accidental redefinition
Exercise 37: Turning weeks into seconds
Exercise 38: Turning seconds into weeks
Exercise 39: Using predefined Java™ standard library constants
Exercise 40: Converting temperature values
Exercise 41: Time unit conversion
Exercise 42: Interest calculation
Exercise 43: Summing short and char

The logical “and” operator &

Boolean “and” of two operands:

```
boolean examSuccess = true,
        registered = false;

boolean examPassed = examSuccess & registered;

System.out.println("Exam passed:" + examPassed);
Exam passed:false
```

Related exercises

Exercise 44: Operator & vs. &&

The += operator

Increment variable by right hand value:

```
int a = 4;

a = a + 2;

System.out.println("Value:" + a);
Value:6
```

```
int a = 4;

a += 2;

System.out.println("Value:" + a);
```

Related exercises

Exercise 45: Strange addition

The &= operator

Logical and operation:

```
boolean examSuccess = true,
        registered = false;

examSuccess = examSuccess & registered;

System.out.println(
  "Exam success:" + examSuccess);
```

```
Exam success:false
```

```
boolean examSuccess = true,
        registered = false;

examSuccess &= registered;

System.out.println(
  "Exam success:" + examSuccess);
```

Arithmetic assignment operators

=	Assign right to left operand
+=	Assign sum of operands to left operand
-=	Assign difference of operands to left operand
*=	Assign product of operands to left operand
/=	Assign quotient of operands to left operand
%=	Assign remainder of operands to left operand

Example using %=

```
int value = 13;
value %= 5;
System.out.println("Result=" + value);
```

```
Result=3
```

Logical assignment operators

&=	Assign logical “and” of operands to left operand
|=	Assign logical “or” of operands to left operand

Related exercises

Exercise 46: Understanding +=

Increment operator ++

Increment variable by 1:	Shorthand version:

`int a = 4;`
 	
`a = a + 1;`
 	
`System.out.println("Value:" + a);`
	`int a = 4;`
	
	`a++;`
	
	`System.out.println("Value:" + a);`

`Value:5`	

Different range behaviour!

Increment variable by 1:	Shorthand version:

`byte value = 127; // Max possible value`
 	
`value = value + 1; // Error: Required type: byte`
`// Provided:int`
 	
`System.out.println(value);`
	`byte value = 127; // Max possible value`
	
	`value++; // o.K., will cycle through range`
	
	`System.out.println(value);`

Does not compile	`Value:-128`

Cast required

Increment variable by 1:	Shorthand version:

`byte value = 127; // Max possible value`
 	
`value = (byte)(value + 1); // cast required,`
`// possible overflow`
 	
`System.out.println(value);`
	`byte value = 127; // Max possible value`
	
	`value++; // cycle through range,`
	`// no cast required.`
	
	`System.out.println(value);`

`Value:-128`	

Prefix and postfix notation

pre-increment		post-increment	

`int a = 4;`
`int b = ++a;`
`System.out.println(b);`
		`int a = 4;`
		`int b = a++;`
		`System.out.println(b);`	

Output:	5	Output:	4

Related exercises

Exercise 47: Three ways expressing the same

Operator examples

Shorthand	Operation	Explicit

`//Integer operations`
`i--;`
`i += k;`
`i %= 3;`
 	
`// boolean`
`  - nothing appropriate -`
	`    Decrement by one`
	`    Raise by k's value`
	`   assign modulus of 3`
	
	`Switching true <--> false`
		`//Integer operations`
		`i = i - 1;`
		`i = i + k;`
		`i = i % 3;`
		
		`// boolean`
		`b = !b;`

Related exercises

Exercise 48: Guessing results
Exercise 49: Cleaning up the mess

Java™ comment flavors

Multi line comment:

```
int a;
/*  We define a variable. Then
    subsequently a value is being assigned */
a = 33;
```

End of line comment:

```
int a;  // We define a variable.
a = 33; // Then subsequently a value is being assigned
```

Inline comments

```
int strength = a /* fixed value */ + b /* age */ + c /* courage */;
```

being run-time equivalent to:

```
int strength = a + b + c;
```

Javadoc™ comments

```
/**
 * Describing rectangles. ❶
 */
public class Rectangle {

  /**
   *
   * @param width Setting the rectangle's new width. ❷
   */

  public void setWidth(double width) {
      // Implementation yet missing
  }
  ...
}
```