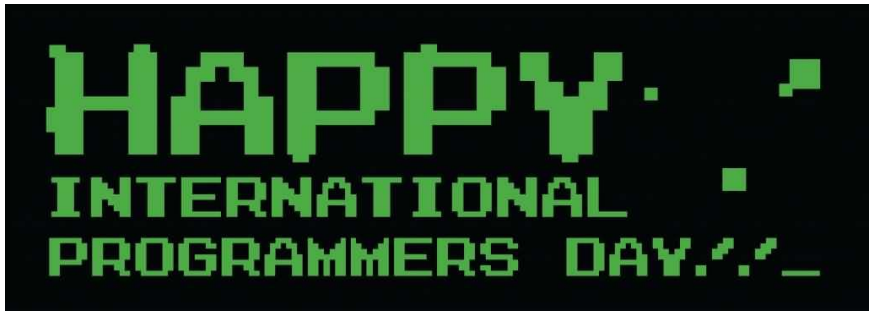
Програмування-1

Лекція 2

Поздоровляю!

- С днем програміста!
- Скоро 13 вересня!
- Це 10000000-й день року!!!
- Якщо хтось не зрозумів, то далі буде ☺



План лекції

- Основи синтаксису
  - Система кодування і пробільні символи
  - Коментарі и JavaDoc-коментарі
  - Зарезервовані слова
  - Ідентифікатори та правила іменування
  - Типи даних та літерали
  - Декларування змінних
- Оператори
  - Пріоритет операторів
  - Арифметичні оператори
  - Логічні оператори
  - Перетворення типів

Основи синтаксису – Кодування

- Unicode
  - UTF-8
    - 1 символ = 1..6 байт
  - UTF-16
    - 1 «звичайний» символ = 2 байта
    - рідкісні символи представляються сурогатними парами
  - UTF-32
    - 1 символ = 4 байта
- Java
  - UTF-16 в пам’яті ВМ
  - UTF-8 на диску (.java и .class- файли)

Обробка Unicode-рядків – складна річ.
Не треба «винаходити велосипед»,
використовуйте відповідні бібліотеки!

Основи синтаксису – Пробільні символи

- Пробіл
- Табуляція
- Розрив рядка

Приклад некрасивої програми:

```
public class HelloWorld {public static void
main (String[]
args){System.out.println("Hello World!");}}
```

Приклад красивої програми:

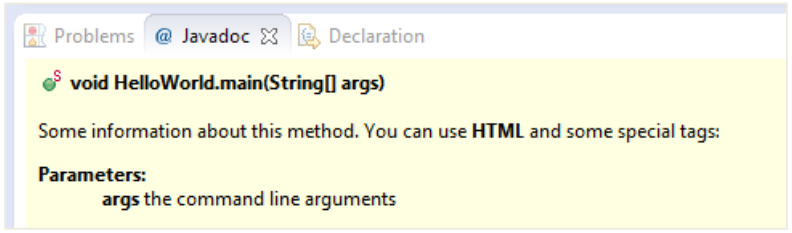
```
public class HelloWorld {
   public static void main (String[] args) {
      System.out.println ("Hello World!");
   }
}
```

Основи синтаксису – Коментарі

- Один рядок // .....
- Декілька рядків /* ..... */
- Javadoc - коментарі /** ..... */

```
    /**
     * Some information about this method.
     * You can use <b>HTML</b> and some special tags:
     * @param args the command line arguments
     */

        // One-line comment
                                                // Another one-line comment
        /* Multi-line comment
        System.out.println("Hello, World!");
        System.out.println("AAAAAAAAAAAAAAAAAAA!");
        */
    }
}
```



Основи синтаксису – Зарезервовані слова

- Ключові слова

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert (1.4)	enum (1.5)	var (1.10)			

- Літерали

true	false	null

Основи синтаксису – Ідентифікатори

- Ідентифікатор
  - може складатись із букв, цифр, символів «$» та «_»
  - не може починатись із цифри
  - має відрізнятись від зарезервованих слів
  - може мати будь-яку довжину
  - може містити будь-які символи *Unicode*
- Не рекомендується
  - використовувати символ $ (цей символ використовується у службових ідентифікаторах)
  - використовувати символ підкреслення
    - (виняток: імена констант, наприклад, MAX_LENGTH)
  - використовувати занадто довгі ідентифікатори
  - використовувати ідентифікатори, назва яких не пояснює їх сенс
  - використовувати символи, що не входять до набору ASCII (знижує легкочитність; ускладнює використання на системах з різними локалізаціями)

- Великі та малі літери розрізняються!
  - Name ≠ name

Основи синтаксису – Правила іменування

- Пакети
  - усі літери маленькі
    - `package helloworld;`
- Класи та інтерфейси
  - с великої літери
  - кожне наступне слово з великої літери
    - `class String {}, class HelloWorld {}, interface Runnable {}`
- Методи, поля, змінні
  - з маленької літери
  - кожне наступне слово з великої літери
    - `public static void main(String[] args) {...}`
    - `public void doSomething() {...}`
    - `int myNewVariable;`
- Константи
  - усі літери великі
  - слова розділяються знаком підкреслення
    - `PI, MAX_LENGTH, MIN_DOUBLE`

Основи синтаксису – Типи даних

- **Примітивні**
  - byte
  - short
  - char
  - int
  - long
  - float
  - double
  - boolean

- **Посилання (на об’єкти)**
  - В тому числі String

Java – суворо типізована мова
- відповідність типів перевіряється на етапі компіляції

В Java відсутня можливість прямого звернення до довільної області пам’яті та адресна арифметика

Основи синтаксису – Примітивні типи даних

Цілі числа				
Тип	Розмір		Область значень	
	байти	біти		
byte	1	8	$-2^7 .. 2^7-1$	-128 .. 127
short	2	16	$-2^{15} .. 2^{15}-1$	-32 768 .. 32 767
int	4	32	$-2^{31} .. 2^{31}-1$	-2 147 483 648 .. 2 147 483 647
long	8	64	$-2^{63} .. 2^{63}-1$	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807
char	2	16	$0 .. 2^{16}-1$	'\u0000' .. '\uffff' (0 .. 65 535)

Дійсні числа			
Тип	Розмір		Область значень
	байти	біти	
float	4	32	$\pm 3.40282347*10^{38} .. \pm 1.40239846*10^{-45}$
double	8	64	$\pm 1.79769313486231570*10^{308} .. \pm 4.94065645841246544*10^{-324}$

Булевий (логічний) тип даних	
Тип	Область значень
boolean	true, false

Основи синтаксису – Посилання

- В Java всі об'єкти знаходяться в пам’яті, що виділяється динамічно (**купа**, **heap**)
  - неможливо створити об’єкт у стеку
  - доступ до об’єктів здійснюється за допомогою посилань (**reference**)

- Спеціальне значення **null** говорить про те, що посилання не пов’язане з жодним об’єктом

Основи синтаксиси – Літерали

- цілі (integer: **int**, **long**)
- дійсні (floating-point: **float**, **double**)
- булеві (**boolean**)
- символи (**char**)
- рядки (**String**)
- null-літерал (**null**)

Основи синтаксису – Цілі літерали

- Десяткова система числення
  - використовується за замовчуванням
  - для чисел **long** в кінці ставлять букву '**l**' або '**L**' (краще '**L**' ☺)
- Шістнадцяткова система числення
  - використовуються символи **0..9, A..F** (або **a..f**)
  - число починається з '**0x**' (або '**0X**')
  - великі чи маленькі букви - значення не має
    - але краще **0xFACE** ніж **0xface**, **0XFACE**, **OXface**
- Вісімкова система числення
  - використовуються символи **0..7**
  - число починається з '**0**'

```
byte b = 12;
short s = 32000;
int n = 100000;
int oct = 0123;  // вісімкова система! ЦЕ НЕ 123 !!!
int hex = 0xCAFE;  // 16-кова система
int hex2 = 0XBeDaBeDa;
int maxInteger = 2147483647;
long badLong =  3000000000;  // Помилка компіляції: integer number too large
long goodLong = 3000000000L; // Помилки немає
long smallL = 30000000001; // букву 'L' легко спутати з одиницею
```

Основи синтаксису – Цілі літерали JDK7+

У версії JDK7 додали:

- Двійкова система числення
  - використовуються символи 0,1
  - число починається з ‘0b’ або ‘0B’
- Роздільник розрядів для дуже довгих чисел
  - в будь-якому місці числа можна вставити символ ‘_’ для покращення читабельності

```
int binary = 0b0101010110101010;

int million = 1_000_000;
long billion = 1_000_000_000L;
long trillion = 1_000_000_000_000L;
```

Короткі теоретичні відомості про системи числення

- Число складається з цифр, які стоять у відповідних розрядах
- Кожен розряд має свій номер
  - починаючи з **0**
- Кожен розряд має свою вагу
- Вага_розряду = Основа_системи_числення у степені номер_розряду
- Значення числа = сума результатів множення цифр числа на вагу відповідних розрядів

Десяткова система числення

Номер розряду	n-1	...	4	3	2	1	0
Вага розряду	10^{n-1}	...	10^4	10^3	10^2	10^1	10^0
Вага розряду	10...000	...	10000	1000	100	10	1
Приклад				**2**	**0**	**1**	**9**

$$2019 = 2*10^3 + 0*10^2 + 1*10^1 + 9*10^0$$
$$= 2*1000 + 0*100 + 1*10 + 9*1$$
$$= 2019$$ ☺

Двійкова система числення

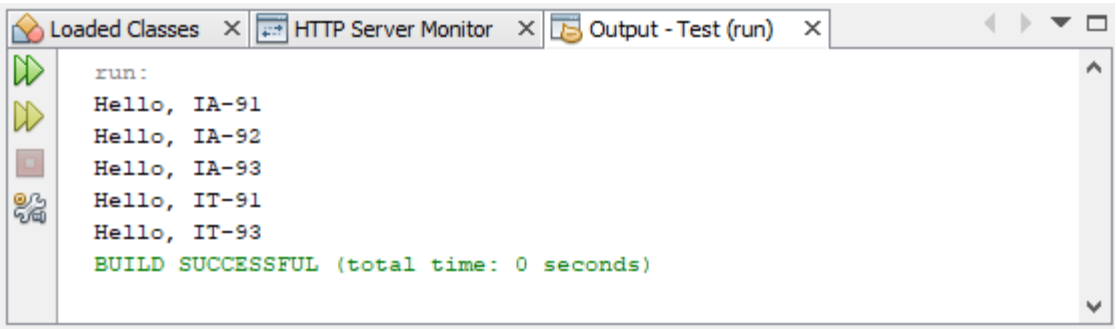
Номер розряду	n-1	…	7	6	5	4	3	2	1	0
Вага розряду	2^{n-1}	…	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Вага розряду	**2^{n-1}**	…	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

91				64		16	8		2	1
92				64		16	8	4		
93				64		16	8	4		1

```
91 = 64 + 16 + 8 + 2 + 1 = 01011011
92 = 64 + 16 + 8 + 4     = 01011100
93 = 64 + 16 + 8 + 4 + 1 = 01011101
```

Двійкова система числення

```
public static void main(String[] args) {
        System.out.println("Hello, IA-" + 91);
        System.out.println("Hello, IA-" + 0b01011100);
        System.out.println("Hello, IA-" + 0x5D);
        System.out.println("Hello, IT-" + 91l);
        System.out.println("Hello, IT-" + 0135);
}
```



Двійкова система числення

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = **10**

Двійкова система числення

- Від’ємні числа представляються у додатковому коді
- Для того щоб з додатного числа зробити від’ємне, його потрібно відняти від 0 ....

0 - 1 = -1

-

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

Двійкова система числення

- ... АБО: інвертувати та додати одиницю
- Приклад: -1

Додатне число (1)	0	0	0	0	0	0	0	1
Його інверсія	1	1	1	1	1	1	1	0
Додамо одиницю ...	0	0	0	0	0	0	0	1
... отримуємо результат (-1):	1	1	1	1	1	1	1	1

Основи синтаксису – Літерали з рухомою комою

- a.k.a. floating-point number
- Використовується десяткова система числення
- За замовченням буде подвійна точність (тип double)
  - для **double** суфікс ‘**d**’/’**D**’ можна не вказувати
  - для **float** потрібно в явному вигляді вказувати ‘**f**’ або ‘**F**’

```
double pi = 3.14159265359; // За замовченням double
double e = 2.71828182846d; // 'd' на конці ні на що не впливає
double speedOfLight = 3e8; // 'науковий формат': = 3 * 10^8
float badFloat = 1.0; // Помилка компиляції cannot convert from double to float
float goodFloat = 1.0f;
```

Основи синтаксису – Булеві літерали

- **true** або **false**
  - обов’язково маленькими літерами!
  - **TRUE** або **False** – помилка!

```
boolean toBeOrNotToBe = true;
boolean toBeAndNotToBe = false;
```

Основи синтаксису – Символьні літерали

- **Один** символ в **одинарних** лапках (приклад: `'a'`)
- «**Особливі**» символи:

\b	\u0008	BackSpace - **BS**
\t	\u0009	Horizontal Tab - **HT** – табуляція
\n	\u000a	LineFeed - **LF** – кінець строки
\f	\u000c	Form Feed - **FF** – кінець сторінки
\r	\u000d	Carriage Return - **CR** – повернення каретки
\"	\u0022	double quote - **"** – лапки
\'	\u0027	single quote - **'** – апостроф
****	\u005c	backslash - **** – зворотна коса риска
\ooo	від \u00**00** до \u00**FF**	будь-який з перших 256 символів, де **ooo** – код символу у вісімковому записі
\uXXXX	від \u**0000** до \u**FFFF**	будь-який **UTF-16** символ, де **XXXX** – код символу у шістнадцятковому записі

Символи є цілими числами типу **char** (0..65535)

```
char b = 'a' + 1;
```

Основи синтаксису – Рядкові літерали

- a.k.a. **String** literals
- **Будь-яка** кількість символів у **подвійних лапках**
- Можуть містити спеціальні символи
  - *дивись попередній слайд*
- Треба пам’ятати:
  - **String** – це **клас**
  - **рядок** – це **об’єкт**
    - до того ж це **immutable** об’єкт

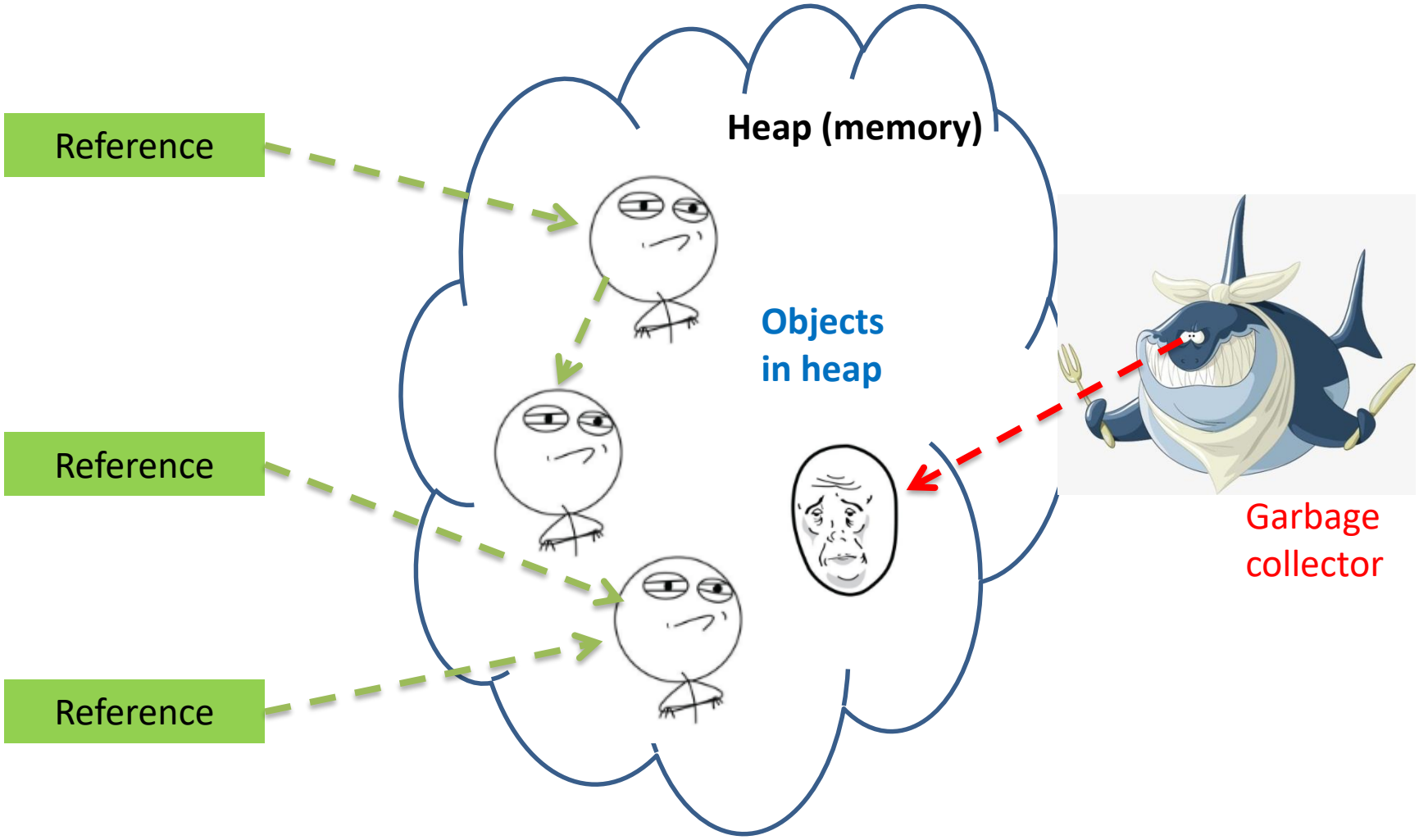
```
String s1 = ""; // рядковий (строковий) літерал нульової довжини (порожній рядок)
String s2 = "Hello, World!"; // неявне створення об’єкта класу String
String s3 = new String ("Hello, World!"); // явне створення об’єкта класу String
```

Основи синтаксису – **null**-літерал

- Використовується для посилань «**ні на що**»
- Не тримайте посилання на непотрібні більше об’єкти. За допомогою присвоєння **null** можна зробити об’єкт недосяжним, щоб **Garbage Collector** очистив пам’ять

```
                String hw = "Hello, World!";
// ...
// ...
                hw = null;
```

Як працює Garbage Collector?



Основи синтаксису – Декларування змінних

- Синтаксис:

[<модифікатори>] <тип> <назва> [= <rvalue>];

```
public static final int tattuqoltuae = 42;
int four = 2*2;
int noInit;
```

- Можна але не бажано декларувати кілька змінних в одному рядку

```
double a,b,c,d,x1,x2; // Не робіть так!
```

- Для локальних змінних не присвоюється значення за замовчуванням
  - Перед тим, як з них щось прочитати, в них потрібно щось записати
  - Компілятор це відслідковує

```
int x;
System.out.println(x); // Помилка компиляції
```

Питання

Скільки буде

2 + 2 = ?

2 * 2 = ?

2 + 2 * 2 = ?

?

Оператори – Пріоритет операторів

Порядок виконання	Оператор
-	. [ ] ( ) (виклик метода)
R to L	++ -- +(унарний) –(унарний) ~ ! (перетворення типу)
L to R	* / %
L to R	+ -
L to R	<< >> >>>
L to R	< > <= >= instanceof
L to R	== !=
L to R	&
L to R	^
L to R	|
L to R	&&
L to R	||
R to L	?:
R to L	= *= /= %= += -= <<= >>= >>>= &= ^= |=

Кілька прикладів

```
int a = 1; // записати 1 в змінну a


a = a + 1; // прочитати a, додати 1, записати в a
a += 1; // збільшити a на 1
a++; // збільшити a на 1


System.out.println(a++); // постінкремент
System.out.println(++a); // преінкремент
```

Оператори – Логічні оператори

- Для **boolean**:

!	&	|	^
Логічне «НІ», NOT	Логічне «І», AND	Логічне «АБО», OR	Додавання за модулем два, XOR

- Для **цілих**:

~	&	|	^
Побітове «НІ» (Доповнення)	Побітове «І»	Побітове «АБО»	Побітове «Додавання за модулем два»

При виконанні цих операторів **завжди** розраховуються обидва операнди

AND, OR, NOT

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

NOT	
0	1
1	0

Питання

Скільки буде

true | true & false = ?

Оператори – Короткозамкнуті логічні оператори

&&	||
Короткозамкнуте логічне «І»	Короткозамкнуте логічне «АБО»

- Правий операнд обчислюється **лише при умові**, що лівого недостатньо
- Часто використовують для запобігання **NullPointerException**:

```
MyDate d;
//...
if ((d != null) && (d.day() > 31)) {
            // do something with d
}
```

Оператори – Конкатенація рядків

```
String name = "Петро";
String surname = "Петренко";
String fullName = surname + " " + name;
```

- Якщо один з операторів String, то інший також перетворюється у String
- Обчислення відбуваються зліва направо:

```
String s1 = "abc" + 2 + 2;   // abc22
String s2 = 2 + 2 + "abc";   // 4abc
```

Оператори – Оператори зсуву

- Арифметичний (знаковий) зсув вправо (>>). При зсуві знаковий біт копіюється
  - 128 >> 1 // 128 / 2^1 = 64
  - 256 >> 4 // 256 / 2^4 = 16
  - -256 >> 4 // -256 / 2^4 = -16
- Логічний зсув вправо (>>>). При зсуві знаковий біт не копіюється, а заповнюється нулями
  - 256 >>> 4 // 16
  - -256 >>>4 // 268435440
- Зсув вліво (<<). Знаковий біт не копіюється
  - 128 << 1 // 128 * 2^1 = 256
  - 16 << 2 // 16 * 2^2 = 64

Оператори – Перетворення типів

- Перед виконанням операцій над даними різних типів, вони приводяться до одного типу

- Перетворення типів буває:
  - Явне (вказується програмістом)
    - Можлива втрата точності
    - Можлива втрата значущості
  - Неявне (виконується компілятором автоматично)
    - Можлива втрата точності

```
long lVal = 1000; // OK. int неявно перетворюється в long
int iVal= lVal; // ПОМИЛКА! long не може бути неявно перетворений в int
int iVal2 = (int)lVal; // Компілюється. Але можлива втрата значущості
```

Оператори – Явне перетворення типів

- Програміст вказує до якого типу потрібно привести:

(новий_тип) rvalue;

- Можлива втрата точності
- Можлива втрата значущості

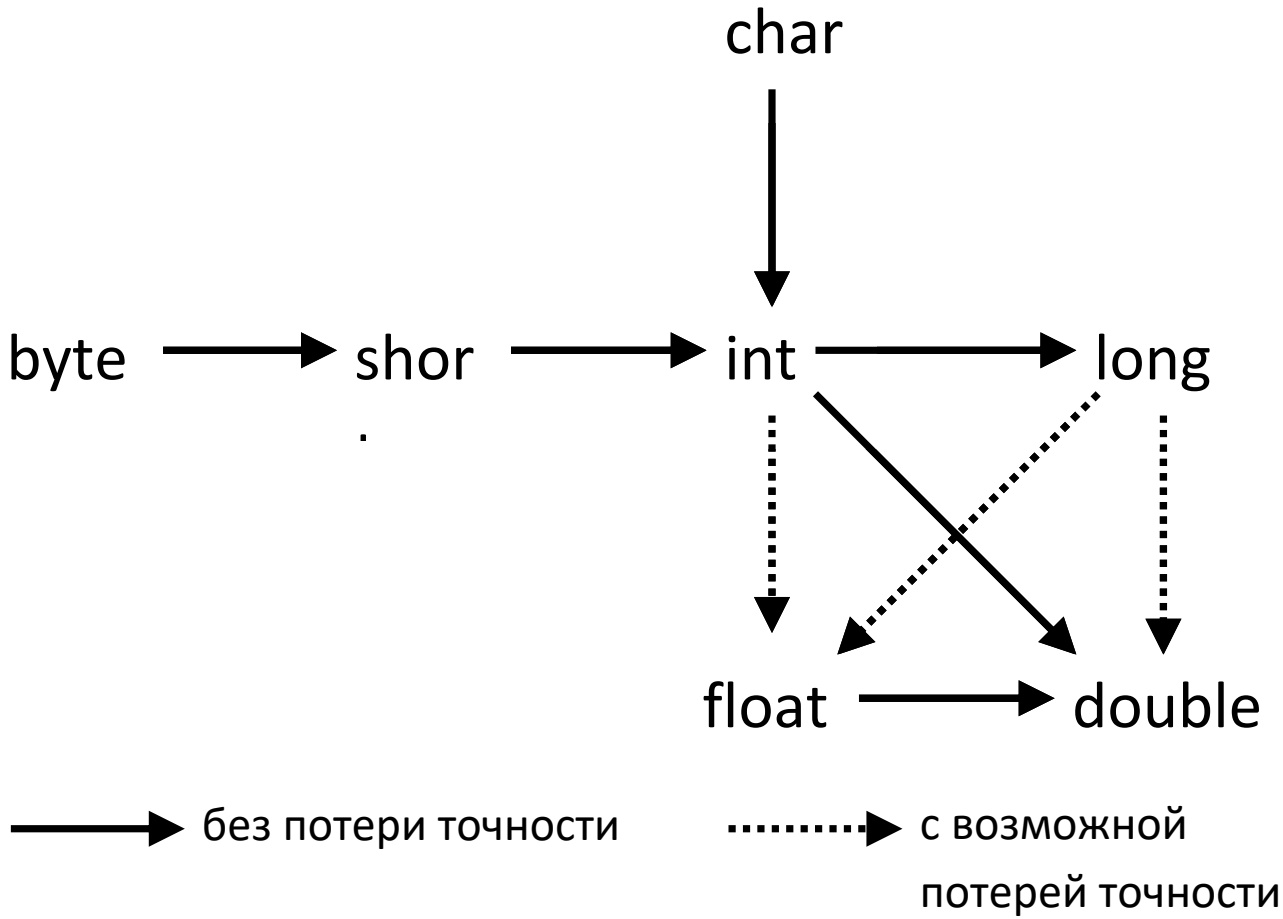
```
double d=3.14159265359;
float f=(float) d; // Втрата точності
System.out.println(f); // 3.1415927

int i = 257;
byte b = (byte) i; // Втрата значущості
System.out.println (b); // 1
```

Оператори – Неявне перетворення типів

Якщо один із операндів має тип	**то інший також перетворюється у**
double	**double**
float	**float**
long	**long**
інакше обидва перетворюються у **int**	

Оператори –
Неявне перетворення типів



Оператори – Приклад 1

```
int chinaPopulation = 1360000000;
int indiaPopulation = 1240000000;

int total = chinaPopulation + indiaPopulation;
System.out.println(total); // ???
```

```
byte b1=1;
byte b2=2;
byte b3=b1+b2; // Помилка компиляції
```

Оператори – Приклад 2

```
int i = 123456789;
float f = i;
double d = i;
short s = (short) i;
System.out.println("int: " + i);
System.out.println("float: " + f);
System.out.println("double: " + d);
System.out.println("short: " + s);
```

Оператори – Приклад 3

```
int a=1;
int b=0;
int c=a/b;
System.out.println(c);
```

Оператори – Приклад 4

```
double a=1;
double b=0;
double c=a/b;

System.out.println(c);
System.out.println("c+1 = " + (c + 1));

System.out.println("+0.0 == -0.0 : " + (0.0 == -0.0));
System.out.println("a/(+0.0) = " + (a/(+0.0)));
System.out.println("a/(-0.0) = " + (a/(-0.0)));
```

Операторы – Приклад 5

```
double a=0;
double b=0;
double c=a/b;

System.out.println("c    =" + c);
System.out.println("c+0  =" + (c + 0));

System.out.println("c<0  =" + (c < 0));
System.out.println("c>0  =" + (c > 0));

System.out.println("c==0 =" + (c == 0));
System.out.println("c!=0 =" + (c != 0));

System.out.println("c==c =" + (c == c)); // :)
System.out.println("c!=c =" + (c != c)); // :)

System.out.println("c == NaN: " + (c == Double.NaN)); // :)))

System.out.println("c is NaN: " + Double.isNaN(c)); // Робіть саме так
```

Клас **Math**

- Math.**sin**()
- Math.**cos**()
- Math.**sqrt**()
- Math.**pow**()
- Math.**log**()
- **…**