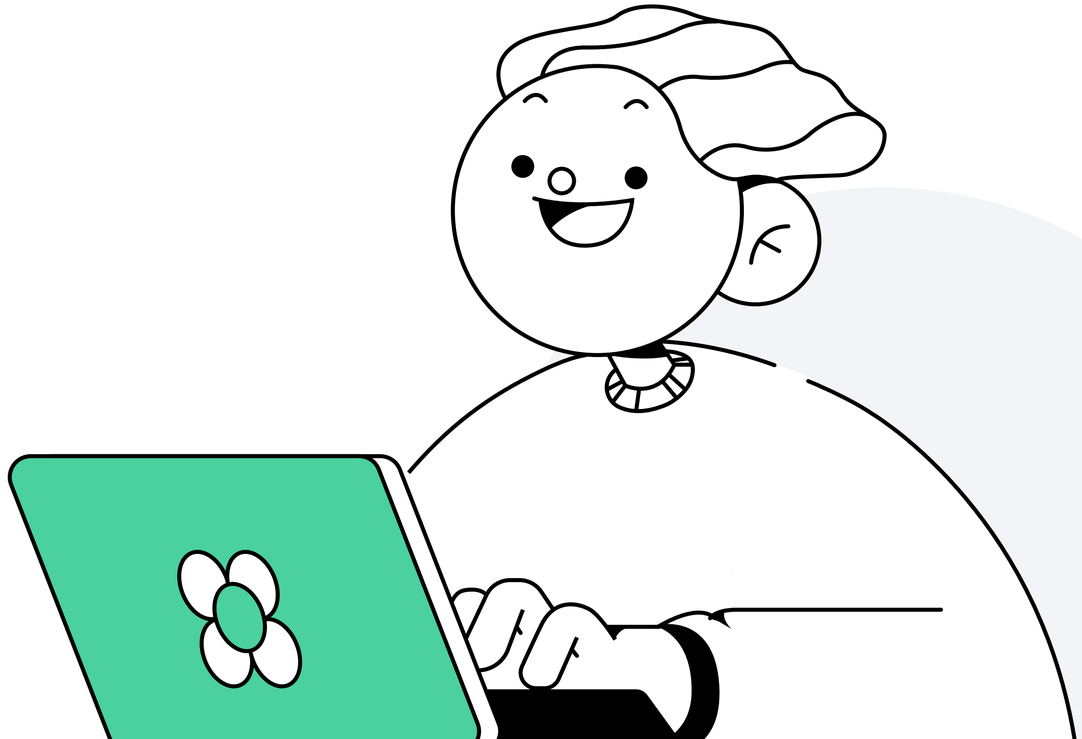
Типы данных Java: примитивы

План занятия

1. Целочисленные типы данных
2. Вещественные типы данных
3. Символьный и логический типы данных



Типы данных в Java

- **Примитивные (Primitive Data Types)**
- Ссылочные (Reference Types)

Примитивные типы

Примитивные типы бывают четырёх видов:

1. Целочисленный
2. Вещественный
3. Символьный
4. Булевый

Целочисленные типы данных

Целочисленные типы

Тип данных	Размер в битах	Диапазон значений
byte	8	от -128 до 127
short	16	от -32,768 до 32,767
int	32	от -2^{31} до $2^{31}-1$
long	64	от -2^{63} до $2^{63}-1$

Представление численных типов в отрицательном виде

Любое число в машинном виде представлено множеством бит, например:

```
цифра 2 выглядит 00000010 (1 байт)
```

Представление численных типов в отрицательном виде

Для представления отрицательных чисел в Java, как и во многих других языках программирования, используется понятие дополнительный код. Чтобы перевести число в дополнительный код, нужно:

1. Инвертировать каждый бит
2. Прибавить к младшему разряду 1

Пример: число 2 в бинарном представлении 00000010

1. Инвертируем каждый bit -> 11111101
2. Прибавим 1 -> 11111110 (дополнительный код)

Дополнительный код используется в вычислительной технике для быстрого вычисления

Подчёркивание чисел

Визуально разряды чисел можно разделять символом подчёркивания _, он не заменяет запятую

Пример:

```
int number1 = 1_234_000;
long number2 = 1_000_000L;
```

Как перекладывать числа из одних переменных в другие

Представим, что типы данных — это коробки:

- **byte** — совсем маленькая коробка
- **short** — коробка побольше
- **int** — коробка среднего размера
- **long** — большого

Если мы захотим переложить число из **int** в **long**, то никаких проблем не возникнет.
Java знает: то, что лежит в средней коробке, легко поместится в большую:

```
int i = 10;
long x = i;
```

Как перекладывать числа из одних переменных в другие

Если мы попросим Java перенести число из **long** в **int**, то какое бы число там ни было, Java будет ругаться, потому что это число потенциально может не влезть в **int**:

```
long x = 10;
int i = x; // строка будет подчёркнута
```

Если мы уверены, что там будет число, которое влезет в **int**, то мы можем попросить Java считать это значение типом **int**:

```
long x = 10;
int i = (int) x;
```

Как перекладывать числа из одних переменных в другие

Если число в переменной **long** действительно не подходит для переменной **int**, а мы при этом просим Java считать его подходящим для **int**, то на консоли мы увидим сломанное число:

```
long x = 10_000_000_000L;
int i = (int) x; // число 10 000 000 000 не будет выведено в консоль
```

Как перекладывать числа из одних переменных в другие

Также ошибка произойдёт, если мы создадим цикл, который будет увеличивать переменную **int** в определённое количество раз и выводить значение на экран.
После того как значение выйдет за пределы допустимого в **int**, Java не предупредит нас и начнёт выдавать поломанные числа:

```
int i = 1;
while (true) {
System out println (i);
i *= 10;
}
```

Операции над целочисленными типами

Над целочисленными типами можно проводить всё те же операции, как в школьной алгебре:

- сложение
- вычитание
- деление
- умножение

Нужно запомнить, что при сложении двух разных типов происходит автоматическое приведение типов к большему. Например, при скложении **byte** и **int**, результат вычисления будет **int**

Пример

Сложение:

```
int a = 10;
long b = 2021L;
long result = a + b; // 2031L
```

Вычитание:

```
int a = 500;
long b = 400L;
long result = a - b; // 100L
```

Сложение byte

Спецификация языка защищает разработчика от переполнения типов, поэтому эта программа не скомпилируется:

```
byte value1 = 120;
byte value2 = 3;
byte value3 = value1 + value2;
```

Вычисления всех примитивных типов меньших **int** автоматически рассчитываются в типе **int**, и результатом их вычисления будет тип **int**:

```
byte value1 = 120;
byte value2 = 3;
int value3 = value1 + value2; // исправим тип на int
```

Сложение int

Важно:

1. Максимальное число, которое можно положить в тип **int = 2147483647**
2. По умолчанию все примитивные типы без литерала являются типом **int**

Поэтому такая программа
не запустится:

```
long value4 = 1_000_000_000;
long value5 = 3_000_000_000;
long value6 = value4 + value5;
```

Чтобы исправить проблему нужно добавить
литерал типа — **L**:

```
long value4 = 1_000_000_000L;
long value5 = 3_000_000_000L;
long value6 = value4 + value5;
```

Вещественные типы данных

Вещественные типы

В Java есть два примитивных типа с плавающей точкой:

Тип данных	Размер в битах	Диапазон значений
float	32	от 1.4e-045 до 3.4e+038
double	64	от 4.9e-324 до 1.8e+308

Используемый стандарт вещественных чисел IEEE Standard 754

Вещественные типы

Float 32 бита — 4 байта

- 23 бита мантисса — около 7 десятичных цифр
- 8 бит экспонента
- 1 бит знаковый

Double 64 бит — 8 байт

- 52 бита используются для мантиссы — около 16 десятичных цифр
- 11 бит экспонента
- 1 бит знаковый

Печать вещественных чисел в консоль

Для вывода вещественных чисел можно использовать метод **System.out.format**, в качестве аргумента нужно указать, сколько символов после запятой оставить для вывода:

```
System.out.format("%.2f", 0.257674);
System.out.format("%.4f", 0.257674);
```

Как запоминать дробные числа в double

Все дробные числа в Java считаются с погрешностями.
Java запоминает и хранит только первые, самые главные, цифры дробного числа. Поэтому если мы попросим Java сравнить на равенство сумму нескольких дробных чисел и равное этой сумме дробное число, то Java не сможет это сделать

Пример:

```
double d1 = 0,3 + 0,3 + 0,3;
double d2 = 0,9;
System.out.println(d1 == d2); // false
System.out.println(d1); // 0,8999999999999
System.out.println(d2); // 0,9
```

Сравнение чисел в double

Если мы хотим сравнивать два числа на равенство, мы всегда должны учитывать погрешность при написании программы.
Порог погрешности выбираем исходя из задачи

Пример:

```
double d1 = 0,3 + 0,3 + 0,3;
double d2 = 0,9;
System.out.println(d1 == d2); // false
System.out.println(d1); // 0,8999999999999
System.out.println(d2); // 0,9

if (Math.abs(d1 - d2) < 0,0000001) {
     System.out.println(“Числа равны”);
} else {
System.out.println(“Числа не равны”);
}
```

Специальные значения в double

При делении на 0, используя переменную **double**, Java воспринимает 0 не в его привычном значении, а как очень маленькое число:

```
double x = 5;
double y = 0;
System.out.println(x / y); // Infinity
```

Соответственно, если мы делим на очень маленькое число, то мы получаем очень большое число, которое Java не может вывести на консоль, но для которого есть специальное значение **Infinity**

Специальные значения в double

Если мы вычитаем одно большое число из другого большого числа, то Java считает, что в результате может получиться любое число:

```
double x = 5;
double y = 0;

double d1 = 5 / y;
double d2 = 4 / y;
System.out.println(d1 - d2); // NaN
```

В этом случае Java вызывает специальное значение **NaN**

Float

По умолчанию Java считает все дробные числа в **double**, поэтому если мы используем переменную **float**, то Java начнёт ругаться.

По аналогии с целочисленными переменными необходимо показать Java, что мы действительно хотим использовать эту переменную, поставив после числа букву **F**:

```
float x = 3.5F;
```

Символьный и логический типы данных

Символьный и логический типы

Тип данных	Размер в битах	Диапазон значений
char	16	1 Unicode символ
boolean	8	true/false

Тип данных char

Тип данных **char** используется для хранения символов.
Всегда один конкретный символ.

Для обозначения символов всегда используются одинарные **кавычки ‘ ‘**. Так Java отличает текст от символа.

Char в основном используются для анализа текста посимвольно

Тип данных char

В Java для char используется кодировка Unicode, а для хранения одного Unicode-символа используется 2 байта (16 бит).

Диапазон допустимых значений — от 0 до 65 536.
Отрицательных значений не существует

Кодировка символов

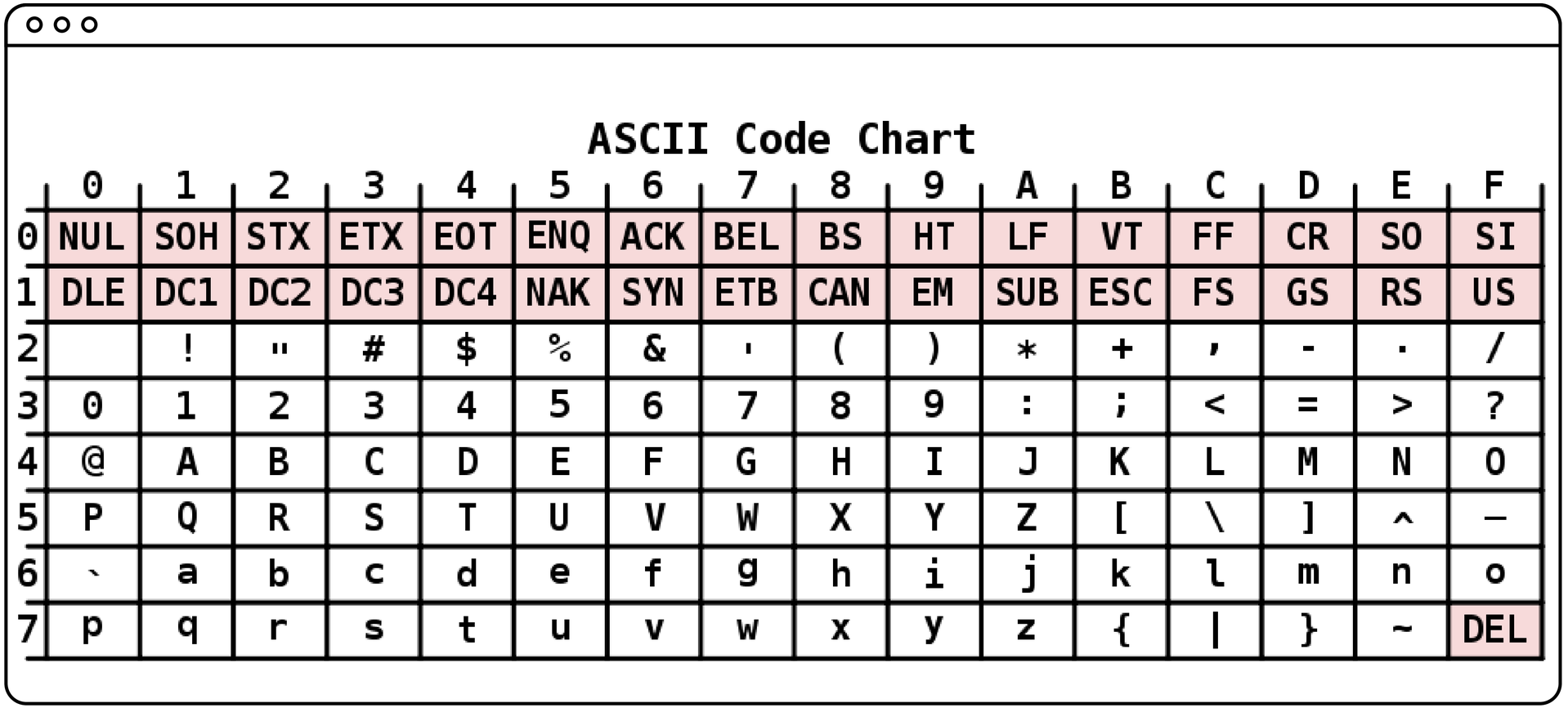
Кодирование символов — это способ представления символов в числовом виде. Зачастую в 8-, 10- или 16-ричной системе счисления.

В Java применяется кодировка UTF-16, в Unix-системах распространена **UTF-8**, в Windows — **CP-1251** или CP-1252.

Все ASCII-символы в UTF-8 занимают 1 байт, остальные — от 2 до 6, чаще 4 байта. UTF позволяет использовать любые символы (хоть китайские), а CP-1251 только ASCII, кириллицу и ещё 62 дополнительных

Тип данных char

Первая таблица ASCII была создана в 1963 году, содержала всего 128 символов и выглядела следующим образом

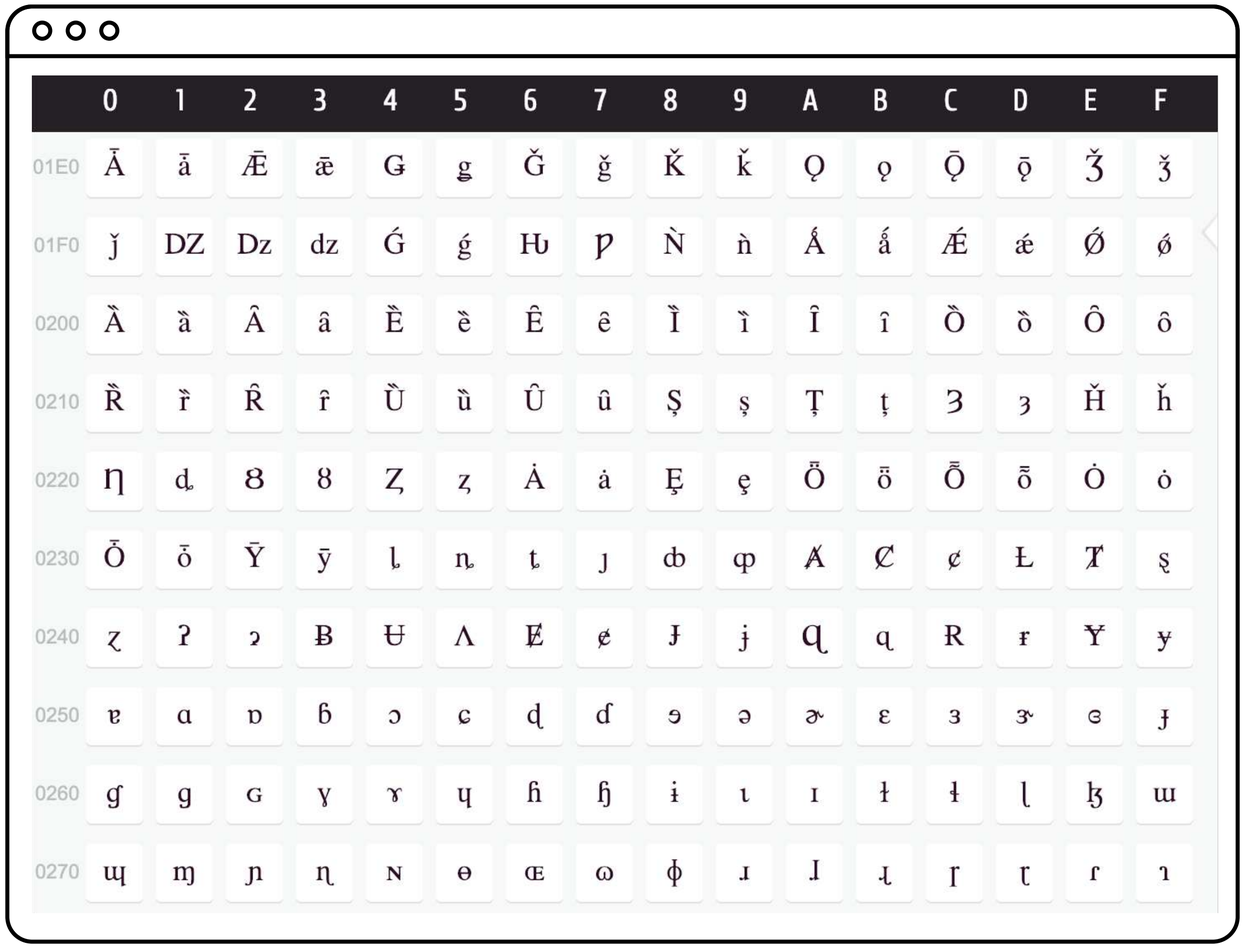


ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Тип данных char

Международная универсальная таблица символов Unicode



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01E0	Ā	ā	Ǣ	ǣ	Ǥ	ǥ	Ǧ	ǧ	Ǩ	ǩ	Ǫ	ǫ	Ǭ	ǭ	Ǯ	ǯ
01F0	ǰ	DZ	Dz	dz	Ǵ	ǵ	Ƕ	Ƿ	Ǹ	ǹ	Ǻ	ǻ	Ǽ	ǽ	Ǿ	ǿ
0200	Ȁ	ȁ	Ȃ	ȃ	Ȅ	ȅ	Ȇ	ȇ	Ȉ	ȉ	Ȋ	ȋ	Ȍ	ȍ	Ȏ	ȏ
0210	Ȑ	ȑ	Ȓ	ȓ	Ȕ	ȕ	Ȗ	ȗ	Ș	ș	Ț	ț	Ȝ	ȝ	Ȟ	ȟ
0220	Ƞ	ȡ	Ȣ	ȣ	Ȥ	ȥ	Ȧ	ȧ	Ȩ	ȩ	Ȫ	ȫ	Ȭ	ȭ	Ȯ	ȯ
0230	Ȱ	ȱ	Ȳ	ȳ	ȴ	ȵ	ȶ	ȷ	ȸ	ȹ	Ⱥ	Ȼ	ȼ	Ł	Ⱦ	ȿ
0240	ɀ	ɂ	ɔ	Ƀ	Ʉ	Ʌ	Ɇ	ɇ	Ɉ	ɉ	Ɋ	ɋ	Ɍ	ɍ	Ɏ	ɏ
0250	ɐ	ɑ	ɒ	ɓ	ɔ	ɕ	ɖ	ɗ	ɘ	ə	ɚ	ɛ	ɜ	ɝ	ɞ	ɟ
0260	ɠ	ɡ	ɢ	ɣ	ɤ	ɥ	ɦ	ɧ	ɨ	ɩ	ɪ	ɫ	ɬ	ɭ	ɮ	ɯ
0270	ɰ	ɱ	ɲ	ɳ	ɴ	ɵ	ɶ	ɷ	ɸ	ɹ	ɺ	ɻ	ɼ	ɽ	ɾ	ɿ

Пример объявления переменной char

```
// символьный литерал
char letter1 = 'N';
// код символа Unicode в десятичном счислении
char letter2 = 78;
// '\uxxxx' - символ Unicode,
// где xxxx цифровой код символа Unicode в шестнадцатеричной форме
char letter3 = '\u0053';
```

Тип данных boolean

Тип данных **boolean** предназначен для хранения логических значений. Логические переменные этого типа могут принимать только два значения: **true** — истина и **false** — ложь

Как выбрать, какой тип использовать в программе

Чтобы выбрать, какой тип использовать, нужно ответить на два вопроса:

1. Какой диапазон значений необходим для переменной?
2. Переменная будет хранить только целочисленные значения?

Использование заведомо больших типов приводит к избыточному использованию ресурсов процессора и оперативной памяти

Значения по умолчанию для примитивных типов

Data Type	**Default Value** (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	‘\u0000’
String (or any object)	null
boolean	false

Полезные ссылки

- Unicode
- Дополнительный код
- Float/Double
- NaN