Рис. 1.1. Рейтинг мов програмування компанії dou.ua за частотою комерційного використання на початок 2018 р. В опитуванні взяло участь 7361 розробників, 90% з яких проживають в Україні. Джерело: https://dou.ua/lenta/articles/language-rating-jan-2018/

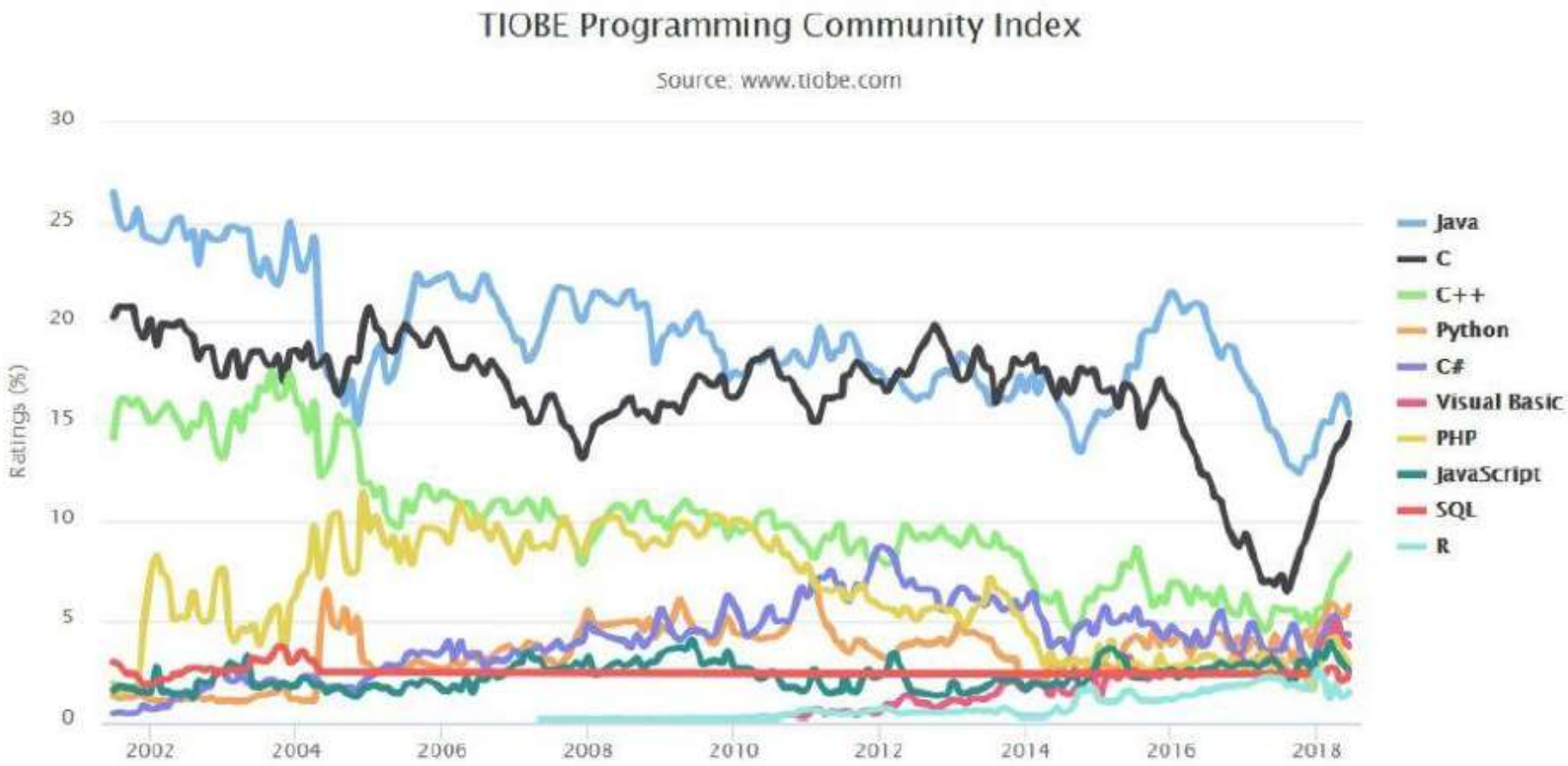

Рис. 1.2. Рейтинг популярності мов програмування в 2002-2018 рр., побудований компанією TIOBE Software шляхом аналізу трендів 25 основних пошукових систем, включаючи Google,Ebay,YouTube, Yahoo !, Wikipedia, Amazon, Bing та ін. Джерело: www.tiobe.com.

Відмінності між мовами C++ та Java

На відміну від C++

В мові Java немає:

1.Вказівників!!!
2.Структур і об'єднань
3.Перевантаження операторів!!!
4.Автоматичного приведення типів з втратою точності
5.Глобальних змінних і функцій
6.Значень аргументів за умовчанням!!!
7.Деструкторів!!!
8.Оператора goto
9.Передачі об'єктів за значенням (тільки за посиланням!)

В мові Java є:

1.Багатопотоковість
2.Пакети замість просторів імен
3.Інтерфейси (аналог абстрактного класу в C++)
4.Вбудований рядковий тип String
5.Документаційний коментар
6.Всі масиви динамічні!

Відмінність властивостей:

1.В C++ true і false – можуть бути числами, в Java – тільки літерали!
2.В C++ специфікатор рівня доступу застосовується до груп полів, в Java – для кожного поля окремо

Java Virtual Machine (JVM) – віртуальна машина Java є основою всієї технології (рис. 1.5). Вона безпосередньо відповідає за виконання java-програм, організовує для них роботу з системними ресурсами (пам'яттю, файловою системою) і системні виклики (тобто організує необхідні виклики функцій конкретної ОС).

Java Runtime Environment (JRE) – це мінімальна реалізація віртуальної машини, необхідна для виконання готового байт-коду. Складається з віртуальної машини – JVM і бібліотеки стандартних Java-класів, але не включає компілятор та інші засоби розробки.

Java Development Kit (JDK) – комплект розробника додатків на мові Java, що включає в себе все, що необхідне для розробки і зборки додатків, в тому числі:

- JRE, яка включає JVM (програма java.exe), стандартні бібліотеки у вигляді файлів .jar та ін.;
- Компілятор Java (програма javac.exe);
- Приклади, документацію;
- Програми-утиліти, що виконують різні допоміжні функції:
  - jar – програма архівації (стиснення) скомпільованих модулів. В результаті стиснення виходить Java-архів (з розширенням .jar), який можна розміщувати на сервері, або запускати на виконання на десктоп-комп'ютері;
  - javadoc – генерує документацію в HTML-форматі з вихідного коду класів, використовуючи для цього спеціальні коментарі в коді;
  - javah – генерує заголовкові та вихідні файли на мові C, за допомогою яких можна використовувати Java-класи в програмах на C;

Eclipse

Eclipse – це вільне інтегроване середовище розробки модульних крос-платформових додатків, яке розвивається і підтримується компанією Eclipse Foundation.

Eclipse відрізняється модульною розширюваною архітектурою, маючи безліч розширень (плагінів) для всіляких завдань розробки. Eclipse також слугує платформою для розробки таких плагінів, чим він і завоював свою популярність: будь-який розробник може розширити Eclipse власними модулями-плагінами. Існують комерційні і некомерційні розширення, найбільш популярними з них є:

- розширення середовища Eclipse для роботи з базами даних, серверами додатків і ін.;
- модулі, що спрощують створення графічних інтерфейсів (наприклад, Window Builder);
- Eclipse JDT (Java Development Tools), який призначений для групової розробки та підтримує системи управління версіями – CVS, GIT;
- розширення для роботи з мовами програмування C/C++, Perl, PHP, JavaScript, Python, Ruby.

Eclipse написана на Java, тому є платформо-незалежним продуктом, тобто може працювати під усіма популярними ОС.

Основними перевагами Eclipse в порівнянні з IntelliJ IDEA є безкоштовність і велика гнучкість за рахунок плагінів. Однак в IDEA набагато краще розвинені засоби інтелектуального аналізу коду. Ці засоби дозволяють, зокрема, при підказці варіантів продовження враховувати весь контекст коду і пропонувати тільки ті варіанти, які підходять і за типом, і за семантикою.

Також IDEA має набагато більше можливостей авторефакторинга і генерації коду. Тому, якщо є можливість платити за ліцензію, зазвичай вибирають IntelliJ IDEA, а Eclipse залишається лідером серед вільних IDE.

NetBeans IDE

Це вільна IDE для мов Java, Python, PHP, JavaScript, C, C++. Проект NetBeans IDE підтримується і спонсорується компанією Oracle, однак розробка NetBeans ведеться незалежною спільнотою розробників-ентузіастів (NetBeans Community) і компанією NetBeans Org.

За якістю і можливостями останні версії NetBeans IDE наближаються до кращих комерційних (платних) IDE, таких як IntelliJ IDEA, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення набраних конструкцій на льоту, безліч вбудованих шаблонів коду. Перевагою цієї IDE також є те, що вже базова версія має повну підтримку технологій Java EE (тобто для розробки веб-додатків не потрібно нічого встановлювати додатково).

Однак в плані зручності розробки NetBeans IDE поступається IntelliJ Idea, оскільки не має настільки розвинених інтелектуальних засобів аналізу коду, а у багатьох задачах (особливо пов'язаних з J2EE-розробкою) вимагає набагато більше ручної роботи зі створення інтерфейсів, дескрипторів та ін.

Особливості синтаксису Java

- Синтаксис Java багато в чому схожий з синтаксисом C# та C++. Як ми побачимо далі, набір ключових слів і операторів цих двох мов мають багато спільного.
- Малі та великі літери – розрізняються (наприклад someName і SomeName – різні імена). Існує угода називати класи з великої літери (SomeClass), змінні та методи – з маленької (someVar, someMethod()), а константи – усіма великими літерами (SOME_CONST).
- Кожен оператор завершується крапкою з комою.
- Оператори можуть перебувати тільки всередині блоків. Блоками є тіло методу, тіло складного оператора (умовного оператора, циклу, секції try, синхронізованої секції), блок ініціалізації класу (про який піде мова в наступних лекціях). Також будь-яка послідовність операторів може бути згрупована в окремий блок. Усюди, де за правилами мови може перебувати одиничний оператор, замість нього може використовуватися блок. У всіх випадках блок виділяється фігурними дужками, наприклад:

```
if (condition) {          // Блок – тіло складного оператора
     operators;
     {                    // Блок – угрупування операторов
          Operators;
     }
}
```

- Ознакою оголошення або виклику методу є круглі дужки після імені:

```
void someMethod() {            // Оголошення методу
     operators;
}
// ...
void someMethod() {            // Оголошення методу
        someMethod();          // Виклик методу
}
```

- Java – повністю *об'єктно-орієнтована* мова. Тобто будь-яка програма будується з класів і об'єктів цих класів, процедурне програмування не підтримується.
- Мова Java *строго типізована*. Це значить, що тип змінної завжди вказується при її оголошенні, а перетворення типів обмежене жорсткими правилами. Наприклад, в Java неможливо перетворити логічне значення в ціле і назад.

Ключові слова (key words)

Нижче наведено список ключових слів Java. Всі ці слова можна використовувати як імена. Жирним виділені ті з них, які є новими в порівнянні з мовою C++. Курсивом виділено ключові слова, зміст яких істотно змінений у порівнянні з C++. Нарешті, сірим виділені зарезервовані слова, які на даний момент не використовуються.

abstract	continue	for	*new*	switch
assert	default	goto	**package**	synchronized
boolean	do	if	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

Оголошення

Всі змінні, методи і класи, використовувані в програмі, повинні бути заздалегідь ***оголошені***. Розглянемо синтаксис оголошень. У дужках [] показані необов'язкові елементи оголошення, а жирним шрифтом виділені обов'язкові ключові слова.

1. Змінні:

```
[Спосіб_доступу] [модифікатори] Тип ім'яЗмінної
                                    [= початковеЗначення];
```

2. Константи:

```
[Спосіб_доступу] static  final  Тип ім'яЗмінної
                                    [= ПочатковеЗначення];
```

3. Методи (функції)

```
[Спосіб_доступу] [модифікатори]
ТипЗначенняЩоПовертається   ім'яМетода (
                          ТипПараметра ім'яПараметра, ...) {
   Оператори;
   return ЗначенняЩоПовертається;
}
```

4. Клас:

```
[Спосіб_доступу] [модифікатори] class им'яКласу {
      Тіло класу (оголошення змінних і методів)
}
```

Тут *Спосіб_доступу* – це один з модифікаторів `private`, `protected`, `public`. Тонкощі використання способів доступу ми розглянемо в наступних лекціях. На початковому етапі спосіб доступу можна просто не вказувати.

До *модифікаторів* змінних відносяться ключові слова `final`, `static`, `volatile`, `transient`. Зміст цих модифікаторів будемо розглядати далі.

Керуючі конструкції

1. Умовний оператор (if)

Синтаксис:

```
if ( <умова> )
        <оператор1>
[else
        <оператор2>]
```

Тут <умова> — це логічний вираз типу boolean, тобто вираз, що повертає true або false. Вирази інших типів, навіть цілочислових, не можуть бути перетворені в boolean, тому перевірка на нуль завжди вимагає явного порівняння:

```
if (a == 0) {...}
```

Як видно з синтаксису, частина else є необов'язковою. Після if і після else стоїть по одному оператору. Якщо потрібно помістити туди кілька операторів, то потрібно поставити **блок** { }. В Java прийнято блок ставити завжди, навіть якщо після if або else стоїть один оператор.

2. Багатоваріантний вибір (switch)

Служить для організації вибору по деякому значенню однієї з кількох гілок виконання.
Синтаксис:

```
switch ( <вираз> ) {
        case <константа1>:
              <оператори1>
        case <константа2>:
              <оператори2>
        ...
        [default:
              <оператори_D>]
}
```

Зауважимо, що <вираз> має видавати ціле число або символьне значення; константи повинні бути того ж типу, що і значення цього виразу.

Елементи case <константа>: є мітками переходу. Якщо значення виразу збігається з константою, то буде здійснено перехід на цю мітку. Якщо значення виразу не збігається ні з однією з констант, то все залежить від наявності фрагмента **default**. Якщо він є, то перехід відбувається на мітку default, якщо його немає, то весь оператор switch пропускається.

В операторі switch фрагменти case не є блоками. Якщо після останнього оператора даного case-фрагмента стоїть наступний case, то виконання буде продовжено, починаючи з першого оператора цього case-фрагмента. Тому в операторі switch зазвичай застосовується оператор **break**, який ставиться в кінці кожного case-фрагмента.

3. Оператори циклу

3.1. Цикл з передумовою (while)

Синтаксис:

```
while ( <умова> )
        <оператор>
```

Як і у випадку оператора if, в Java прийнято <оператор> заключати в фігурні дужки. Приклад:

```
int x = 123, i=0;
while ( x>0 ) {
    x /= 2;
    i++;
}
```

3.2. Цикл с постумовою (do while)

Синтаксис:

```
do
        <оператор>
while ( <умова> );
```

Відрізняється від попереднього тільки тим, що умова кожен раз перевіряється не перед, а після виконання тіла циклу. Тому в **do while()** тіло циклу завжди виконується як мінімум один раз, в той час як **while()** може не виконатися жодного разу, якщо умова є помилковою спочатку.

Приклад:

```
int x = 123, i=0;
do {
    x /= 2;
    i++;
} while ( x>0 );
```

Даний приклад еквівалентний попередньому.

3.3. Цикл for()

Синтаксис:

```
for ( <ініціалізація>; <умова>; <зміна> )
     <оператор>
```

Вираз <ініціалізація> виконується один раз перед першим витком циклу. Перед кожним витком перевіряється <умова>, і після кожного витка циклу виконується вираз <зміна>. Як і в if() вираз «умова» може бути тільки логічним (типу boolean).

Для зручності складання циклів з лічильником в даній конструкції можливо кілька варіантів:

- <ініціалізація> може бути не виразом, а описом змінної з ініціалізацією, наприклад `int i = 0`. Така змінна є *локальною* для циклу і не доступна після його завершення.
- <ініціалізація> може бути списком виразів через кому, наприклад `i = 0, r = 1`.
- <зміна> також може бути списком виразів, наприклад `i++, r *= 2`. Ці два випадки використання оператора «*кома*» є винятком. Ніде, крім заголовка циклу for цей оператор не використовується.
- Всі складові (<ініціалізація>, <умова> і <зміна>) є необов'язковими. Якщо відсутня умова, то вона вважається завжди істинною, і вихід з циклу повинен бути організований якимись засобами всередині самого циклу.

Приклад:

```
int x = 123;
for (int i = 0; x>0; i++) {
x /=2;
}
```

3.4. Цикл for each

Синтаксис:

```
for (<оголошення змінної> : <масив або колекція> )
     <оператор>
```

Це дуже зручна модифікація циклу for, яка дозволяє пройти по всіх елементах масиву або колекції без використання лічильника. Рекомендується всюди, де це можливо, використовувати саме таку форму циклу. Усередині цієї форми for обов'язково оголошується нова локальна змінна, тип якої повинен бути сумісний з типом елементів масиву (колекції). Наприклад:

```
int sum =0, a[] = {5,6,7,8};


  for(int i : a)  sum+=i;
```

Тут перед циклом оголошується масив `a[]` з чотирьох цілих чисел. Всередині циклу оголошується змінна `i` – теж ціла. На кожному витку виконання циклу в змінну `i` заноситься (копіюється) значення чергового елемента з масиву, і для неї виконується тіло циклу. Цикл гарантовано обробить всі елементи масиву і після цього завершиться. Зверніть увагу, що в даному випадку через змінну `i` неможливо змінити вміст масиву. Зовсім інша ситуація буде, якщо в масиві зберігаються не числа, а об'єкти – до цього питання повернемося в наступній лекції.

4. Оператори переходу: break, continue, return.

Використання цих операторів повністю аналогічно мові C++.

break – завершує виконання циклу або оператора switch();

continue – перехід на наступний виток циклу;

return – завершення поточного методу з можливістю повернути значення в метод, що його викликав.

Як приклад спільного використання операторів розгалуження і циклів розглянемо програму, яка генерує випадковим чином 100 символів латинського алфавіту і класифікує їх як "голосні", "приголосні" і "іноді голосні". В останню категорію віднесені символи 'y' і 'w'.

```
public class SymbolTest {

    public static void main(String[] args) {
        for ( int i = 0; i < 100; i++ ) {
            char c = (char)(Math.random()*26 + 'a');
            System.out.print(c + ": ");
            switch ( c ) {
            case 'a': case 'e': case 'i':
            case 'o': case 'u':
                System.out.println(" - голосна");
                break;
            case 'y': case 'w':
                System.out.println(" - інколи голосна");
                break;
            default:
                System.out.println(" - приголосна");
            }
        }
    }
}
```

Введення-виведення

У цій лекції розглянемо засоби *консольного введення-виведення*. Самі по собі консольні додатки використовуються нечасто, але вивчені класи і методи дозволять в майбутньому аналогічно виконувати введення-виведення з текстовими файлами і іншими текстовими потоками, наприклад, з мережевими з'єднаннями.

Форматоване виведення за допомогою об'єкта `System.out`

Крім методів print() і println(), з якими ми вже знайомі, об'єкт System.out надає методи format() і printf() для форматованого виведення. Можливості цих методів однакові і збігаються з можливостями функції printf() мови C.

Формат виведення вказується за допомогою першого параметру – рядка, що містить *специфікатори формату*. Для цілих чисел використовуються специфікатори %d, %x; для дійсних – %f, %e, %g (незалежно від типу – float або double); для рядків – %s. Після символу % можна вказувати ширину поля виведення і кількість знаків після коми для речових значень. Наприклад:

```
double d = Math.PI;
System.out.printf("%f; %10.2f;\n %-10.0ff\n", d, d, d);
```

виведе

```
3,141593;       3,14;
 3        f
```

Якщо тип переданого значення не відповідає типу специфікатора, то генерується *виключення*. Якщо таке виключення не оброблено явно, то це приводить до аварійного завершення програми.

Для форматованого виведення в рядок можна використовувати статичний метод `String.format()`, який має аналогічні параметри. Він створює об'єкт класу String, і в нього записує відформатований текст, наприклад:

```
System.out.println( String.format("%s = %f", "Pi", Math.PI) );
```

Клас `java.util.Scanner`

В Java існує кілька способів реалізації форматованого введення. Найбільш простий з них – використовувати клас `java.util.Scanner`. Цей клас знаходиться в пакеті java.util, тому його необхідно підключити на початку програми:

```
import java.util.Scanner;  // Підключаємо клас на початку програми
```

При створенні об'єкт класа Scanner зв'язується з потоком введення, файлом чи рядком:

```
Scanner in = new Scanner(System.in);          // Стандартний потік введення
Scanner in1 = new Scanner( "123 + 125" );          // Рядок
Scanner in2 = new Scanner( new File("a.txt") );    // Файл в директорії
                                                   // проекту
```

У разі читання з файлу додатково необхідно обробляти можливі *виключення*. У всіх випадках вхідні дані розглядаються як послідовність *слів*, між якими стоять *роздільники*. За замовчанням роздільниками є пробіл, табуляція і новий рядок. Для читання даних за словами в класі Scanner є методи виду **next*Type***(), де *Type* – один з примітивних типів:

```
String s = in.next();    // зчитати наступне слово у вигляді рядка
int a = in.nextInt();    // зчитуємо ціле число
byte b = in.nextByte(); // зчитуємо байтове число
...
double d = in.nextDouble();  //зчитуємо дійсне число
```

Розглянемо тепер приклад програми, яка не просто виводить текст, але також вводить вихідні дані і проводить над ними деякі дії. Нехай необхідно порахувати суму всіх натуральних чисел від *a* до *b*. На Java це буде виглядати так:

```
import java.util.Scanner;                  // Підключаємо  клас із бібліотеки

public class Sum {                       // Основний клас програми

     public static void main(String[] args) {        // Головний метод

        Scanner scanner = new Scanner(System.in);  // Створюємо об'єкт для читання
        System.out.println("Enter a and b");        // Виводимо текст
        int a = scanner.nextInt();       // Читаємо два цілих числа
        int b = scanner.nextInt();
        int res = 0;                     // Змінна для збереження результату
        for (int i = a; i<b; i++) {      // Зчитуємо суму
            res += i;
        }
        System.out.println("The sum is "+res);     }
}
```