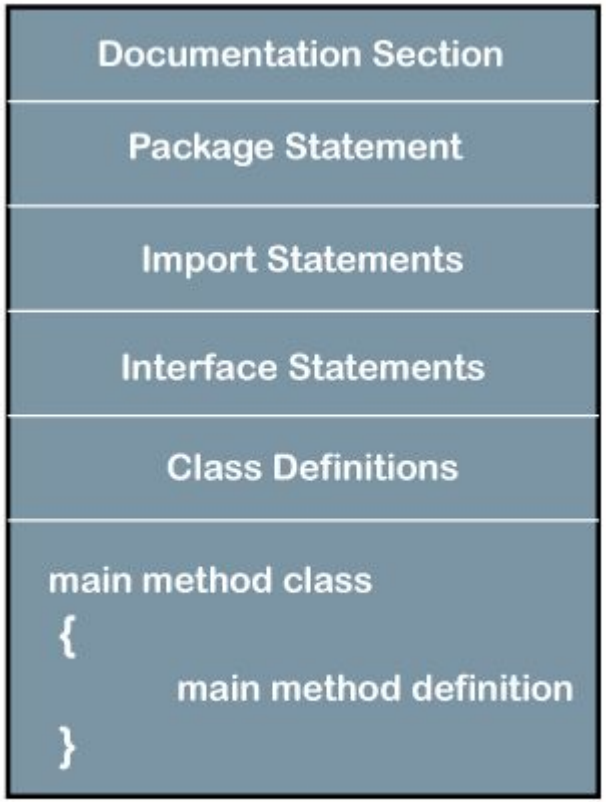
Lesson 2

15.12.2025

Структура класу Java



Structure of Java Program

```
package org.allyourcode.myfirstproject;
```

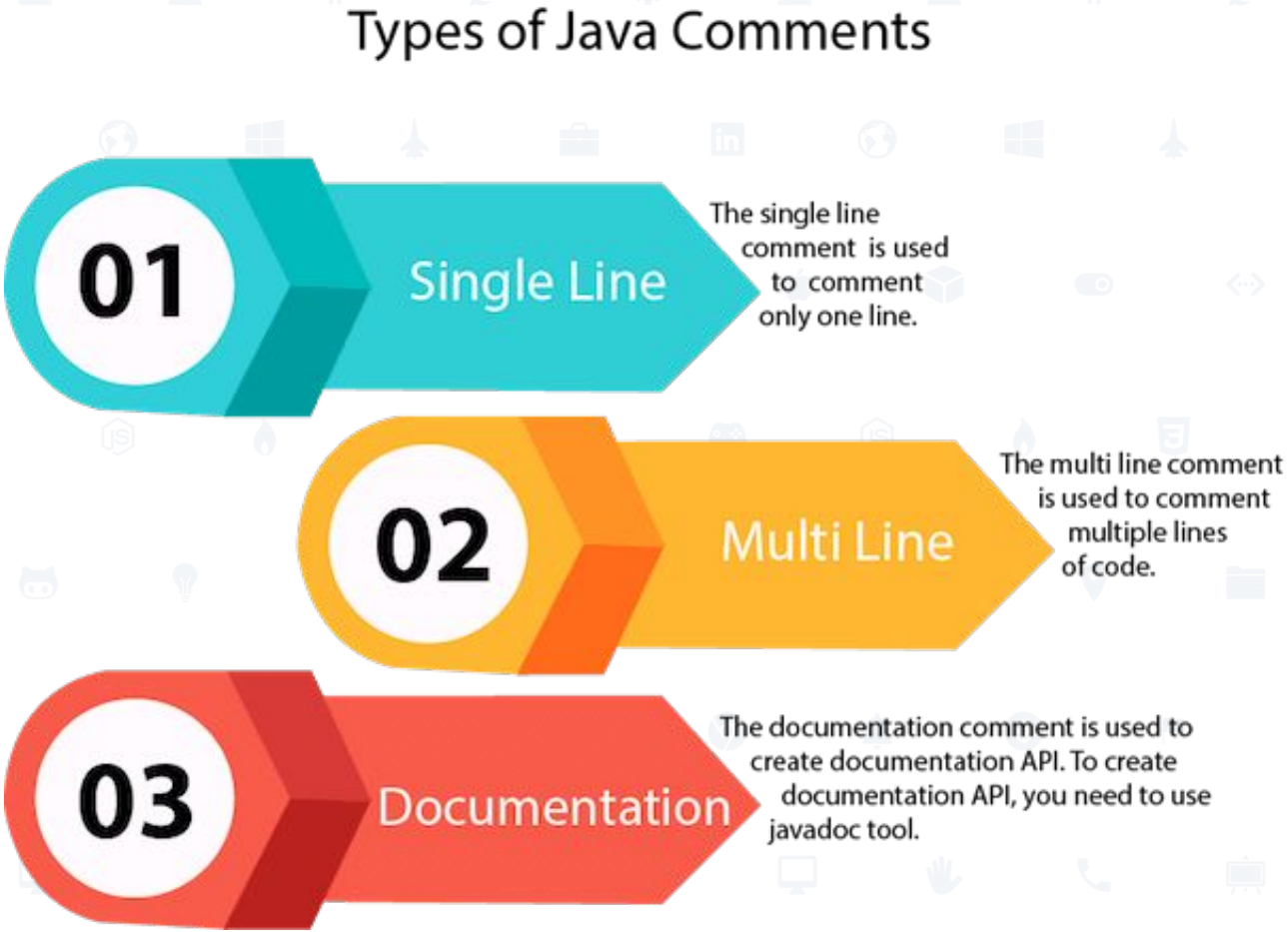
```
public class MyFirstJavaClass {

  /**
   * @param args
   */
  public static void main (String[] args) {
    javax.swing.JOptionPane.showMessageDialog
                                        (null, “Hello”);

  }

}
```

Комментарі


Types of Java Comments
01
Single Line
The single line comment is used to comment only one line.
02
Multi Line
The multi line comment is used to comment multiple lines of code.
03
Documentation
The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

Класс vs. Файл

```
// Compiling this Java program would
// result in multiple class files.

class Sample
{

}

// Class Declaration
class Student
{

}
// Class Declaration
class Test
{
        public static void main(String[] args)
        {
            System.out.println("Class File Structure");
        }
}
```

main()

```
1 | public static void main(String[] args) { }
```

public – модифікатор доступу

static – ключове слово, яке показує що не треба створювати екземпляр класу для виклику методу

void – значення, що повертається (нічого не повертати)

main – ім'я методу, яке дозволяє JVM ідентифікувати початкову точку запуску програми

String[] args – масив параметрів, з якими може запускатися програма

Імена змінних та методів

1. Ім'я має складатися із латинських букв. Хоч джава і підтримує кирилицю — все ж таки краще називати змінні латиницею. Від себе додам давайте їм англійські назви.

2. Назва має нести певний сенс. Це означає, що змінні з ім'ям ***a*** або ***newVariable*** не буде нести ніякого сенсу. Постарайтеся, щоб ім'я було «розмовляючим». Це нелегка праця — вигадувати імена для змінних, класів та методів. Згодом та з досвідом робити це буде легше.

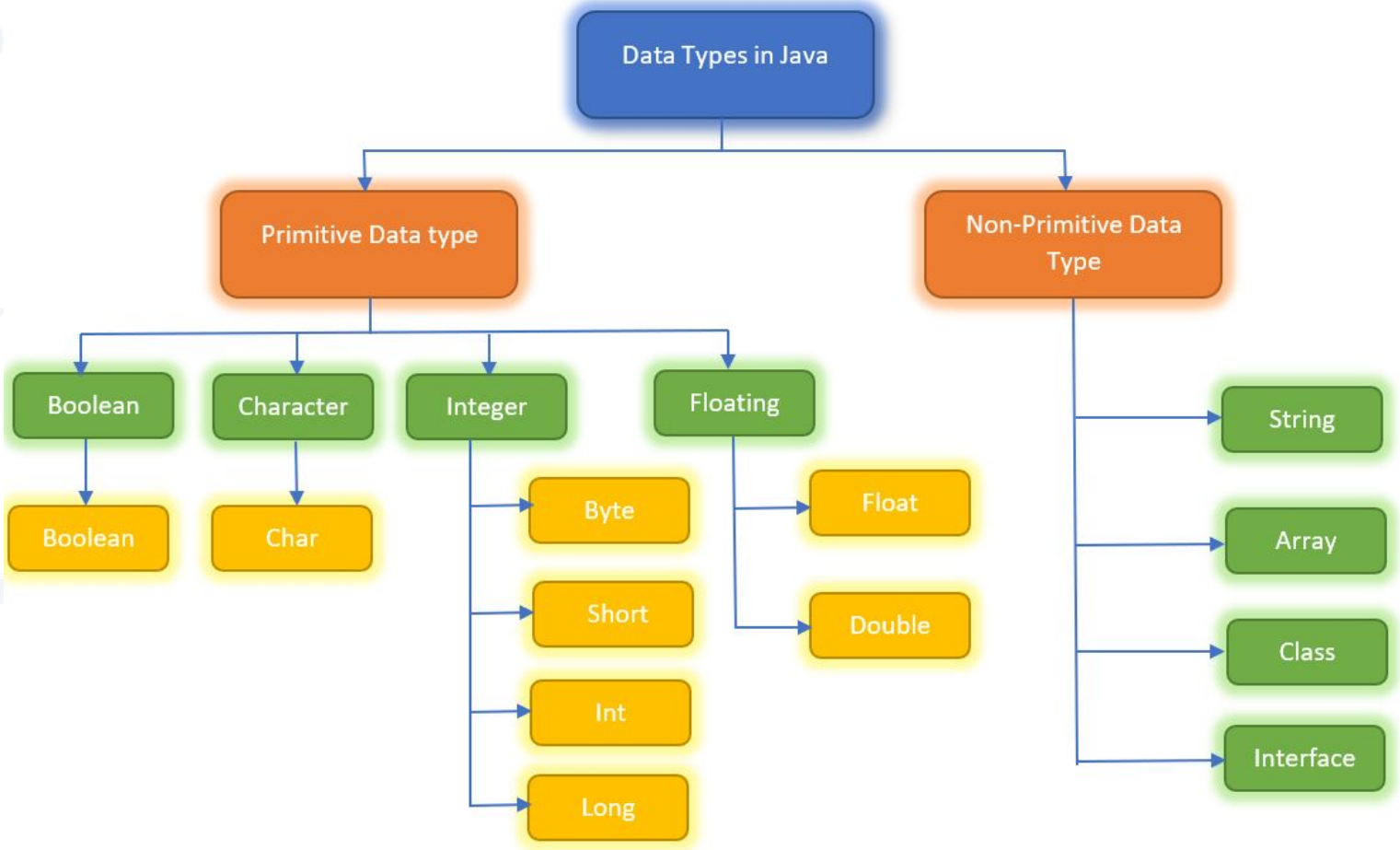
3. Не можна використовувати цифри на початку імені. Це не тільки негласне правило, а й правило мови java. Якщо Ви спробуєте використовувати цифри на початку імені – програма не скомпілюється. Якщо ж цифри в середині або в кінці імені - то їх використання припустимо якщо ж цифра нестиме якийсь сенс змінної, а чи не просто ***a1***, ***a2***.

4. Хоча використання символу ***$*** і _ допустимо на початку назви, я б порадив Вам по можливості уникати ці символи в іменах

5. Програмісти, які пишуть програми мовою java використовують **camelCase** (верблюжий стиль). Якщо ім'я змінної складається більше, ніж із одного слова Наступне слово пишеться з великої літери: userName, jackpotFunctionalView.

6. Не можна використовувати зарезервовані слова як ім'я. На малюнку Нижче представлені всі службові 54 слова в мові java. Вчити та запам'ятовувати їх не потрібно. Згодом вони самі «засядуть» у голові.

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


Data Types in Java
Primitive Data type
Non-Primitive Data Type
Boolean
Character
Integer
Floating
String
Array
Class
Interface
Boolean
Char
Byte
Short
Int
Long
Float
Double

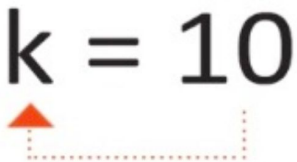
Java's primitive types

Ordered by descending upper range limit

CATEGORIZATION	NAME	UPPER RANGE	LOWER RANGE	BITS
Floating point	double	Really huge positive	Really huge negative	64
	float	Huge positive	Huge negative	32
Integral	long	9,223,372,036,854,775,807	-9,223,372,036,854,775,808	64
	int	2,147,483,647	-2,147,483,648	32
	char	65,535	0	16
	short	32,767	-32,768	16
	byte	127	-128	8
Boolean	boolean	No numeric equivalent	true or false	1

Створення та ініціалізація змінної

1. За допомогою операції надання знака = ми присвоюємо значення змінної. Причому вона завжди відпрацьовує праворуч-наліво:



Значення 10 присвоюється змінною k

2. Надавати значення змінної ми можемо двома способами: в 2 рядки або в 1 рядок

```
1  class Test {
2      public static void main(String args[]){
3          int k;
4          k = 10;
5          System.out.println (k);
6      }
7  }
```

```
1  class Test {
2      public static void main(String args[]){
3          int k = 10;
4          System.out.println (k);
5      }
6  }
```

3. Також необхідно запам'ятати:

Якщо привласнюємо значення змінної типу String, укладаємо його у подвійні лапки

String title = "Java";

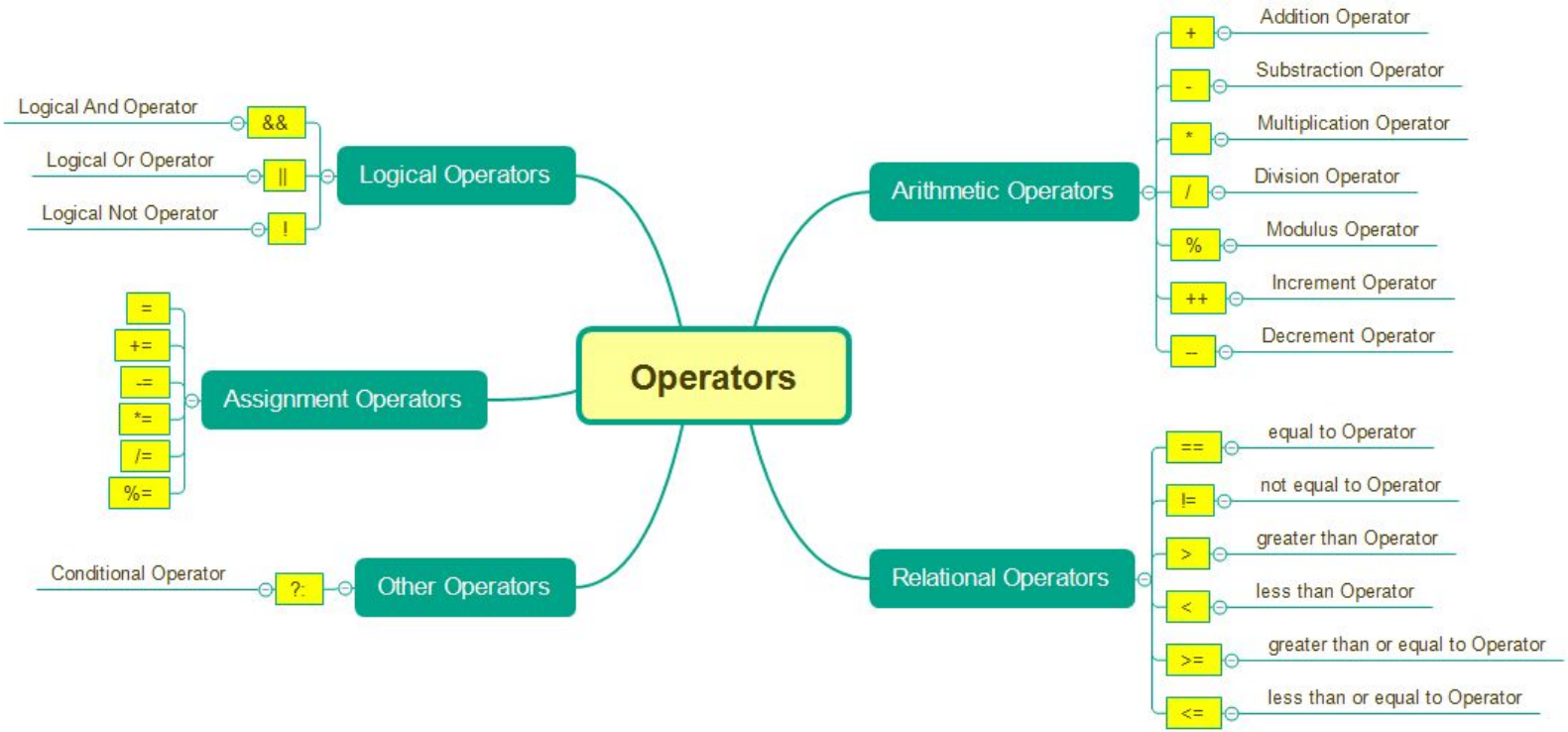
Якщо надаємо значення змінної типу char, укладаємо його в одинарні лапки

char letter = 'M';

Якщо надаємо значення змінної типу float, обов'язково після нього додаємо букву f

float pi = 3.14f;

4. Слово "ініціалізація" - це те саме, що і "привласнити початкове" значення змінної"


Operators
Logical Operators
Logical And Operator
&&
Logical Or Operator
||
Logical Not Operator
!
Assignment Operators
=
+=
-=
*=
/=
%=
Other Operators
Conditional Operator
?:
Arithmetic Operators
+
Addition Operator
-
Substraction Operator
*
Multiplication Operator
/
Division Operator
%
Modulus Operator
++
Increment Operator
--
Decrement Operator
Relational Operators
==
equal to Operator
!=
not equal to Operator
>
greater than Operator
<
less than Operator
>=
greater than or equal to Operator
<=
less than or equal to Operator

Інкремент та декремент

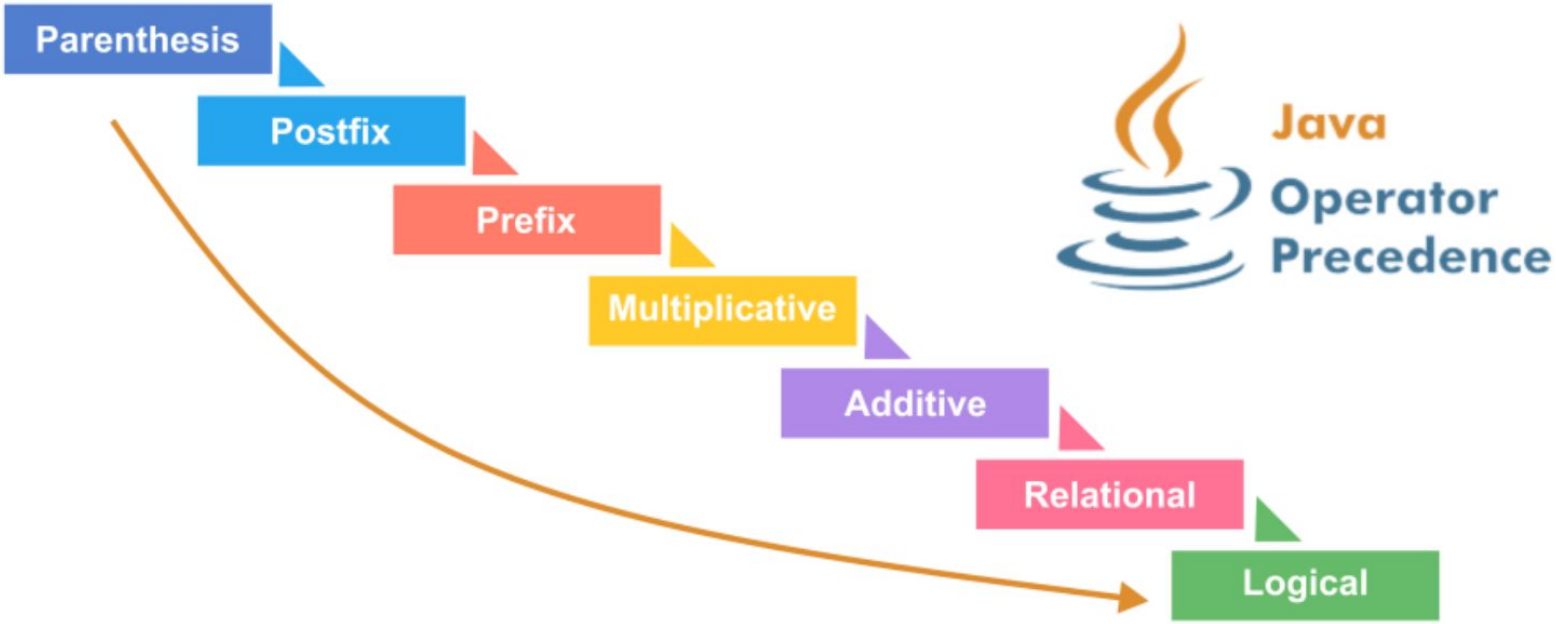
У програмуванні дуже часто доводиться виконувати операції, коли:

змінна має збільшитися на одиницю à +1
змінна має зменшитися на одиницю à -1

Тому вигадали окремі операції зі змінними, які називаються інкремент та декремент.
Інкремент – відповідає за збільшення змінної на одиницю. Позначається як **++**. Наприклад, якщо у нас є змінна i і ми до неї застосуємо інкремент, тоді це буде записано як i++. А це означає, що значення змінної i має бути збільшено на 1
Декремент – відповідає за зменшення змінної на одиницю. Позначається як **--**. Наприклад, якщо у нас є змінна n і ми до неї застосуємо декремент, тоді це буде записано як n-. А це означає, що значення змінної n має бути зменшено на 1.

Тож у чому ж відмінність між постфіксною та префіксною формами?
У постфіксній формі: спочатку використовується старе значення у обчисленнях далі у наступних обчисленнях використовується вже нове значення
У префіксній формі: відразу використовується нове значення у обчисленнях

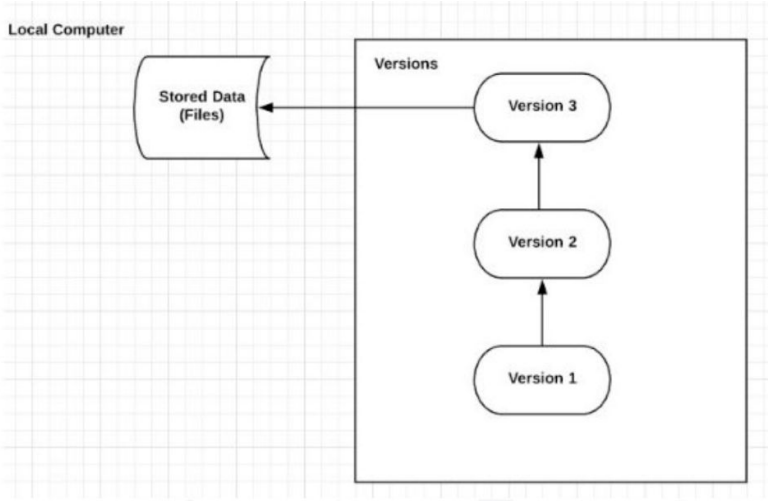
Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to

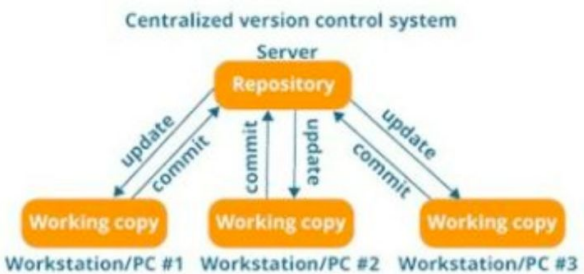

Parenthesis
Postfix
Prefix
Multiplicative
Additive
Relational
Logical
Java
Operator
Precedence

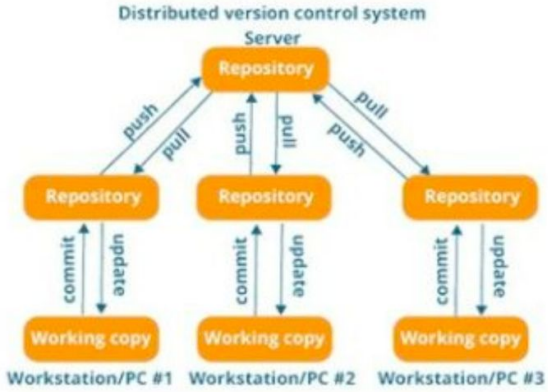
Operators	Notation	Precedence/Priority
Postfix	expr++ , expr--	1
Unary	++expr , --expr , +expr –expr , ~ , !	2
Multiplicative	* , / , %	3
Additive	+ , -	4
Shift	<< , >> , >>>	5
Relational	< , > , <= , >= , instanceof	6
Equality	== , !=	7
Bitwise AND	&	8
Bitwise Exclusive OR	^	9
Bitwise Inclusive OR	|	10
Logical AND	&&	11
Logical OR	||	12
Ternary	? :	13
Assignment	= , += , -= , *= , /= , %= , &= , ^= , |= , <<= , >>= , >>>=	14

Types of Version Control System

- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System







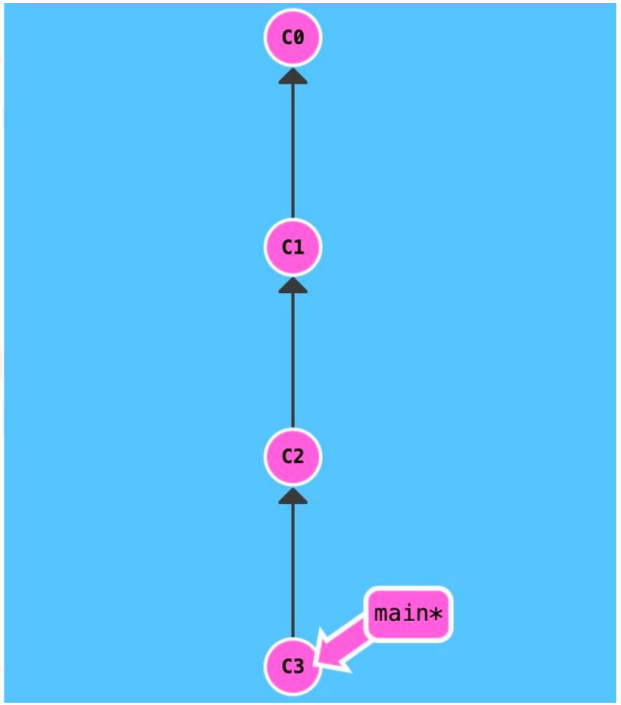
Git – розподілена система керування версіями. Проект був створений Лінусом Торвальдсом для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року

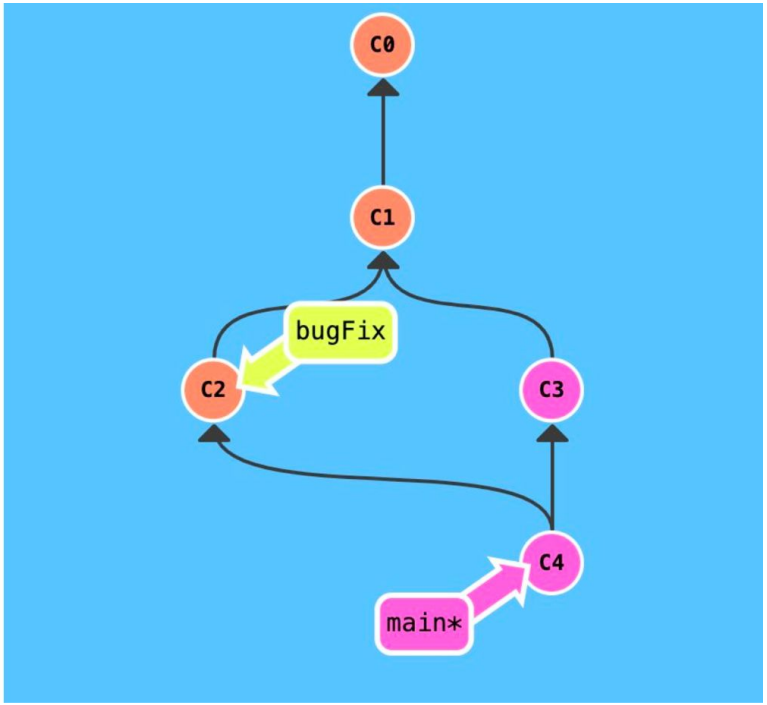
Bitbucket

GitLab

Як працює

Якщо подивитися на картинку, то ставати трохи простіше з розумінням. Кожен гурток, це commit. Стрілки показують напрямок, з якого commit зроблений наступний. Наприклад C3 зроблено з C2 і т. д. Всі ці commit знаходяться у гілці під назвою main. Це основна гілка, найчастіше її називають master. Прямокутник main* показує в якому commit ми зараз знаходимося, вказівник.





Налаштування

Ви встановили собі Git та можете ним користуватися. Давайте тепер його налаштуємо, щоб, коли ви створювали commit, вказувався автор, хто його створив.

Відкриваємо термінал (Linux та MacOS) або консоль (Windows) та вводимо наступні команди.

```
#Установим имя для вашего пользователя
#Вместо <ваше_имя> можно ввести, например, Grisha_Popov
#Кавычки оставляем
git config --global user.name "<ваше_имя>"

#Теперь установим email. Принцип тот же.
git config --global user.email "<адрес_почты@email.com>"
```

Створення репозиторію

Тепер ви готові працювати з Git локально на комп'ютері.

Створимо наш перший репозиторій. Для цього пройдіть до папки вашого проекту.

```
#Для Linux и MacOS путь может выглядеть так /Users/UserName/Desktop/MyProject
#Для Windows например C://MyProject
cd <путь_к_вашему_проекту>

#Инициализация/создание репозитория
git init
```

Тепер Git відстежує зміни файлів вашого проекту. Але, тому що ви тільки створили репозиторій у ньому немає коду. Для цього потрібно створити commit.

```
#Добавим все файлы проекта в нам будующий commit
git add .
#Или так
git add --all

#Если хотим добавить конкретный файл то можно так
git add <имя_файла>

#Теперь создаем commit. Обязательно указываем комментарий.
#И не забываем про кавычки
git commit -m "<комментарий>"
```

Гілка – це набір commit, які йдуть один за одним. Гілка має назву, основну гілку найчастіше називають master / main. Якщо говорити простими словами, то гілка master / main – це наш проект. Інші гілки – це окреме місце для реалізації нового функціоналу або виправлення багів (помилок) нашого проекту. Тобто, з окремою гілкою ви робите будь-що, а потім зливаєте ці зміни в основну гілку master / main.

Не рекомендую створювати commit безпосередньо в master / main. Краще для цього заводити нову гілку та всі зміни писати там.

При створенні нової гілки, намагайтеся називати її коротким і ємним ім'ям. Щоб одразу було зрозуміло, що саме змінювалося за проектом. Якщо ви використовуєте якусь систему для ведення завдань, то можете на початку назви гілки вказувати ID завдання, щоб можна було легко знайти, на основі якого завдання було створено гілку.

Для того, щоб створити нову гілку, вводимо:

```
git branch <название_ветки>

#или вот так

git checkout -b <название_ветки>
```

Перемикатися між гілками можна такою командою:

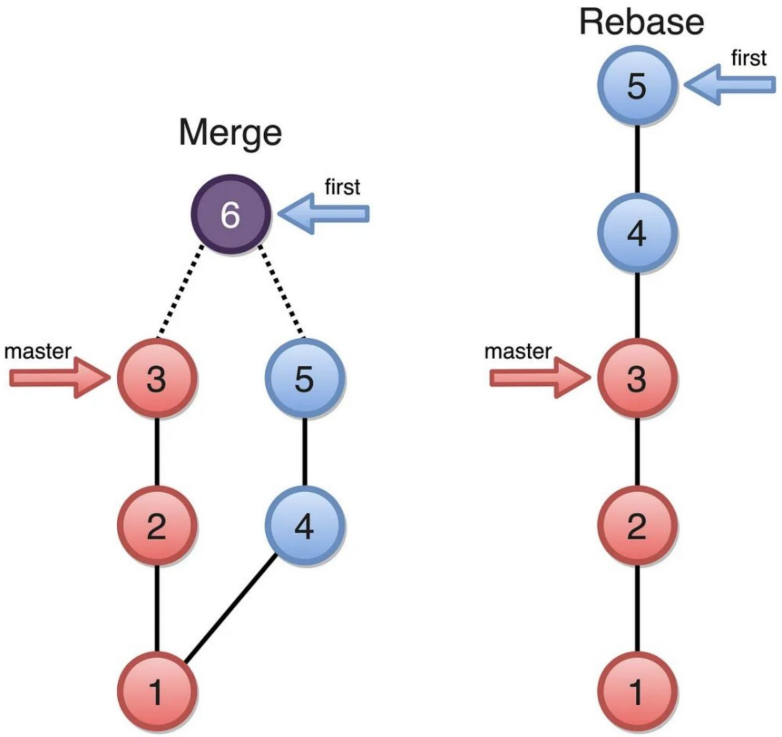
```
git checkout <название_ветки>
```

Після того, як ви завершили роботу над своїм завданням, гілку можна злити в master. Для цього потрібно перейти у гілку master і виконати наступну команду:

```
# Переключаемся в master
git checkout master
# Обновляем локальную ветку с сервера
git pull origin master

# Делаем merge вашей ветки, в ветку в которой вы находитесь
# В данном примере это master
git merge <название_ветки>
```

Rebase (перебазування) - один із способів у git, що дозволяє об'єднати зміни двох гілок. У цього способу є перевага перед merge (злиття) - він дозволяє переписати історію гілки, надавши той історії той вид, який нам потрібний.



Команда git cherry-pick бере зміни, що вносяться одним коммітом, і намагається повторно застосувати їх у вигляді нового комміту у поточній гілці. Ця можливість корисна в ситуації, коли потрібно забрати парочку коммітів з іншої гілки, а не зливати гілку повністю з усіма внесеними до неї змінами.

Команда **git commit --amend** - це зручний спосіб змінити останній коміт. Вона дозволяє об'єднати проіндексовані зміни із попереднім коммітом без створення нового комміту. Її можна використовувати для редагування коментаря до попереднього коміту без зміни стану коду в ньому.

Команда **git squash** це прийом, який допомагає взяти серію коммітів і ущільнити її. Наприклад, припустимо: у вас є серія з N коммітів і ви можете шляхом стиснення перетворити її на один-єдиний комміт. Стиснення через git squash в основному застосовується, щоб перетворити велику кількість малозначимих коммітів на невелике число значних. Так легше відстежувати історію Git.

Команда **git reset** – це складний універсальний інструмент для скасування змін. Вона має три основні форми виклику, що відповідають аргументам командного рядка --soft, --mixed, --hard. Кожен із цих трьох аргументів відповідає трьом внутрішнім механізмам керування станом Git: дереву коммітів (HEAD), розділу проіндексованих файлів та робочого каталогу.

Команда **git stash** - ця команда використовується для збереження непідтверджених змін окреме сховище, щоб можна було повернутися до них пізніше. Самі файли повертаються до вихідного стану. Команда корисна, коли ви працюєте над однією гілкою, хочете перейти на іншу, але ви ще не готові зробити коміт у поточній гілці. Таким чином, ви ховаєте зміни в коді, перемикаєтеся на іншу гілку, повертаєтеся до вихідної гілки, а потім розархівуєте свої зміни.