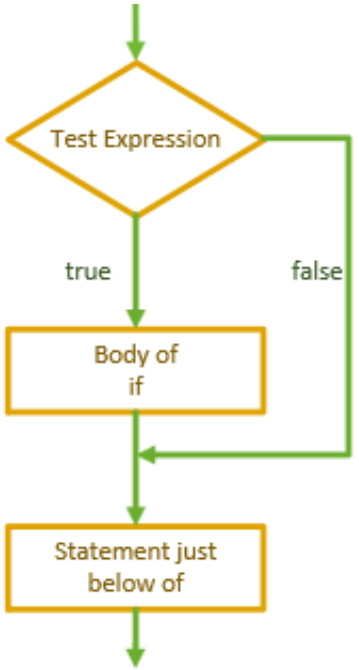
Лекція 4. Засади ООП в Java

1. Розгалуження в Java
2. Реалізація циклічних алгоритмів
3. Класи, об’єкти та посилальний тип даних
4. Реалізація ООП в Java
5. Структура класу, поля та методи
6. Інкапсуляція, доступ до членів класу
7. Конструктор, створення об’єктів
8. Використання пакетів

if statement



if statement is used to test a condition and decide the execution of a block of statements based on that condition result.
If the condition is **true**, the block of statements is executed;
if it is **false**, then the block of statements is ignored.

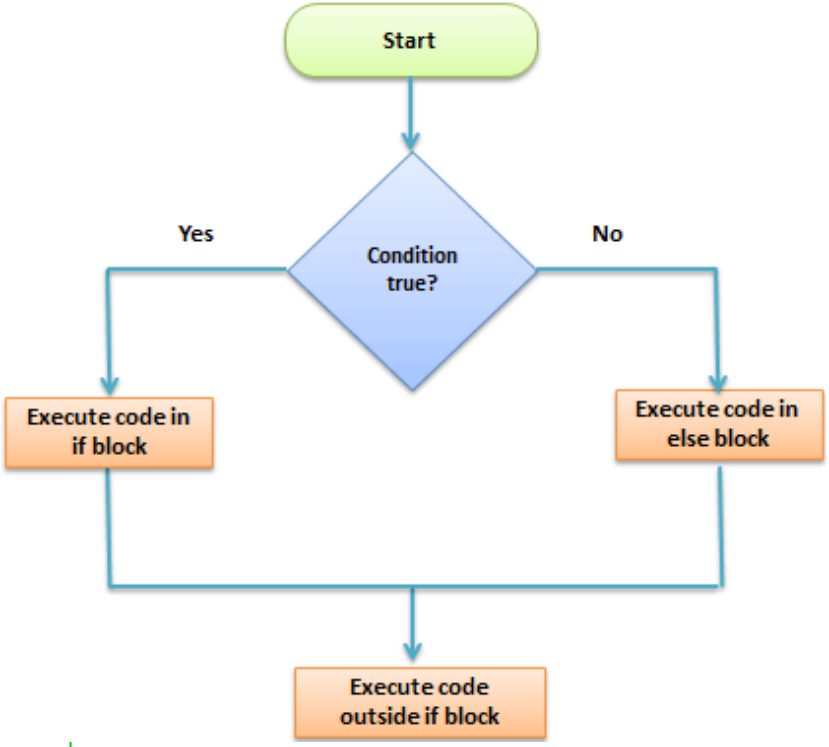
```
if (condition) {
    then-statement;
}
```

```
double mark = 4.3;

if (mark > 3) {
    System.out.println("You passed the exam");
}
```

if-else statement

i***f-else*** statement is used to run a specific block of a program if the condition is true or else

```
if (condition) {
    then-statement;
} else {
    else-statement;
}
```




```
if (temperature < 10) {
      System.out.println("It's too cold");
} else {
     System.out.println("It's Ok"); ;
}
```

Ternary Operator ? :

Ternary (conditional operator) consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable

```
condition ? value1 : value2
```

What are the values of variable str1 and str2?

```
int t = 5, s = 4;
String str1 = (t >= ++s) ? "yes" : "no";

int a = 3, b = 2;
String str2 = a-- == b ? "yes" : "no";
```

switch statement

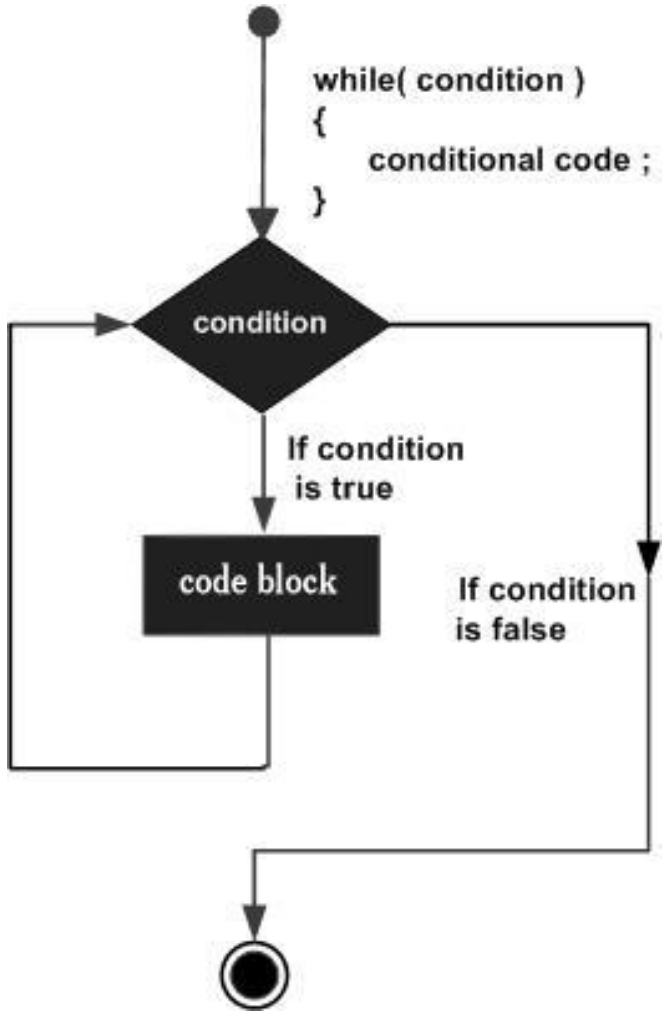
The ***switch*** statement selects one of many code blocks to be executed

The value of the expression is compared with the constant of each case. If there is a match, the associated block of code is executed.

The break and default keywords are optional

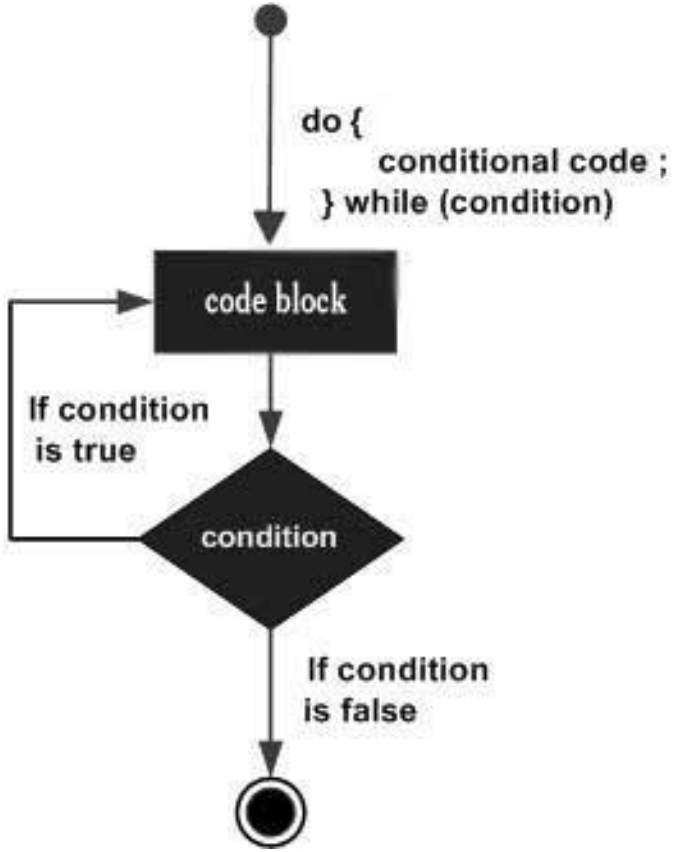
```
switch (expression)
{
      case const-expr1 :
            statement(s);
            break;
      case const-expr1 :
            statement(s);
            break;
      default :
            statement(s);
            break;
}
```

While loops




```
while (condition){
    statements;
}
```

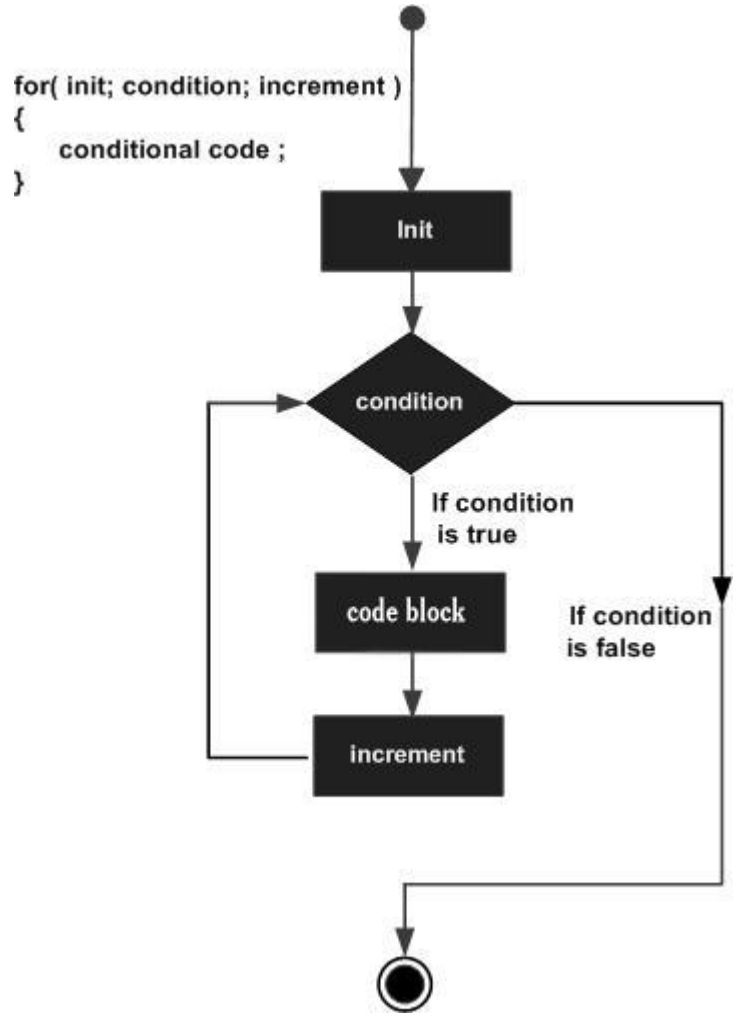
```
do {
    statements;
} while (condition);
```



for statements

```
for (init; condition; increment) {
    statement(s);
}
```

```
for (type variable : collection) {
    statement(s);
}
```



The **break** statement terminates the for, while and do-while loop

```
Scanner sc = new Scanner(System.in);

int n = 0;
for (int i = 0; i < 5; i++) {
    System.out.println("Input number");
    n = sc.nextInt();
    if (n < 0){
        break;
    }
}
System.out.println(n);
sc.close();
```

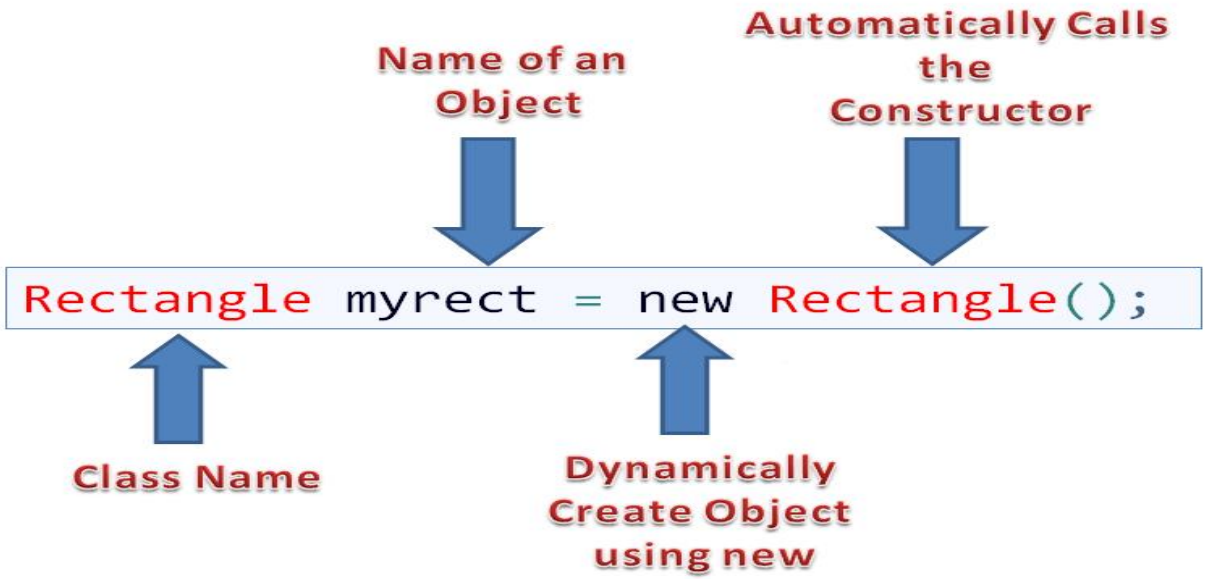
```
Scanner sc = new Scanner(System.in);
int sum = 0;
int n;
for (int i = 0; i < 5; i++) {
    System.out.println("Input number");
    n = sc.nextInt();
    if (n < 0){
        continue;
    }
    sum += n;
}
System.out.println(sum);
sc.close();
```

The **continue** statement skips the current iteration the **for**, **while** and **do-while** loop

REFERENCE TYPES

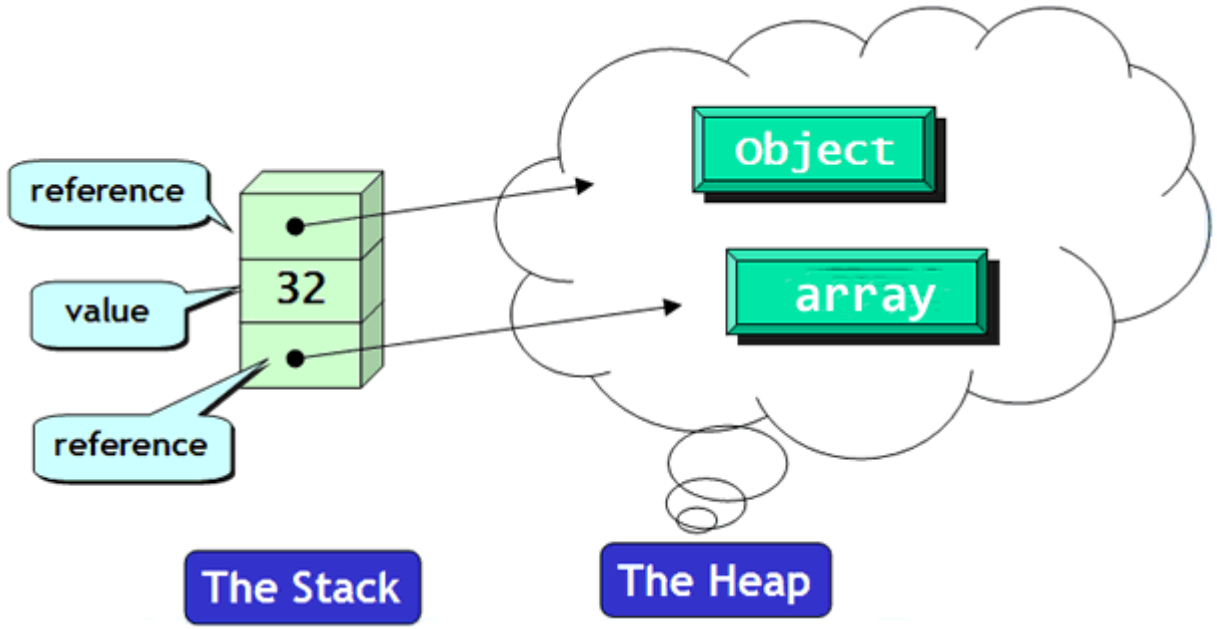
This statement has ***three*** parts:

- Declaring a Variable to refer to an Object;
- Instantiating a Class using **new** operator;
- Initializing an Object using the Constructor.



Reference Types

- A **Reference** is a data element that ***holds the address of a memory location***.
- A **Reference Variable** contains a "**pointer**" to an object.



Class vs Object

Student

has

- Last name
- First name
- Age
- List of courses

can

- Pass an exam
- Enroll to course

student1

Last name - Petrenko

First name - Ostap

Age - 19

List of courses – Java, MQC

student2

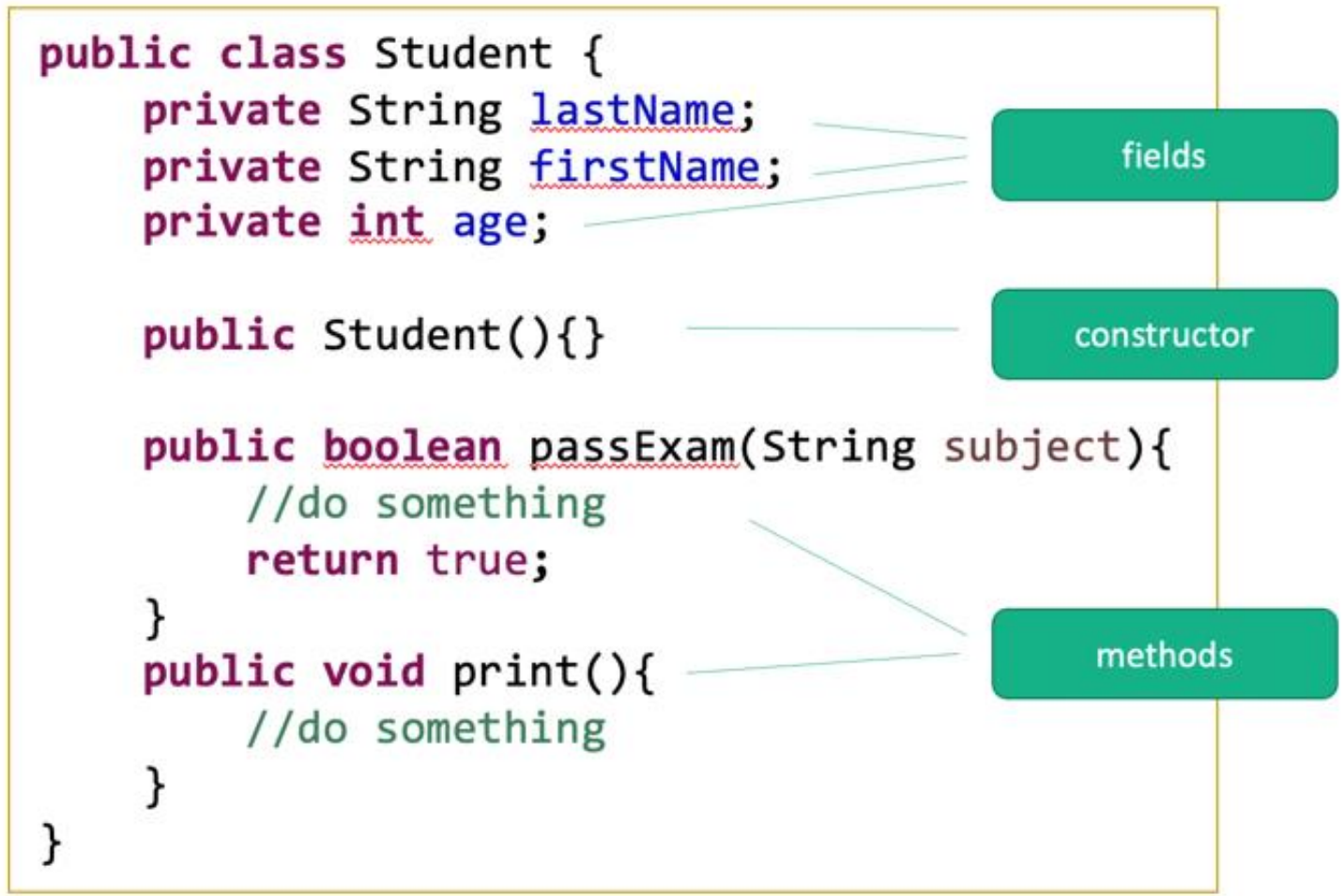
Last name - Romaniv

First name - Maryna

Age - 21

List of courses – Java, ATQC

Клас


public class Student {
private String lastName;
private String firstName;
private int age;
public Student(){}
public boolean passExam(String subject){
//do something
return true;
}
public void print(){
//do something
}
}
fields
constructor
methods

ОСНОВИ JAVA ООП

Концепції ООП ("три кити"):

- **інкапсуляція** (encapsulation) - концепція побудови класів через закриття(капсулювання) їхньої реалізації.
- **успадкування** (inheritance) - створення одних класів на основі інших
- **поліморфізм** (polymorphism) - можливість використання батьківських класів замість класів нащадків. По суті є частиною реалізованої в мові концепції успадкування.

ОСНОВИ JAVA ООП

Згідно принципу інкапсуляції намагаються, щоб доступ до полів екземпляру здійснювався лише за допомогою певних методів.

Об’єкт – це конкретна реалізація певного класу. На основі одного класу може бути створено безліч об’єктів. При цьому в об’єктах виділяють:

- Поведінку об’єкту – що можна зробити з даним об’єктом, або які методи можна застосовувати до нього
- Стан об’єкту – те як об’єкт змінюється, коли Ви застосовуєте його методи
- Ідентичність об’єкту – відмінність об’єкту від інших об’єктів. Об’єкти можуть мати однаковий стан, проте все рівно вони ідентифікуються як різні об’єкти.

Fields

```
public class Student {
   public String name;
   public int age;
   ...
}
```


```
Student stud = new Student();
stud.name = "Krystyna";
stud.age = 22;
```

Accessors

```
Student student = new Student();
```

set

```
student.setName("Oleg ");
```

get

```
String nameStud = student.getName();
```

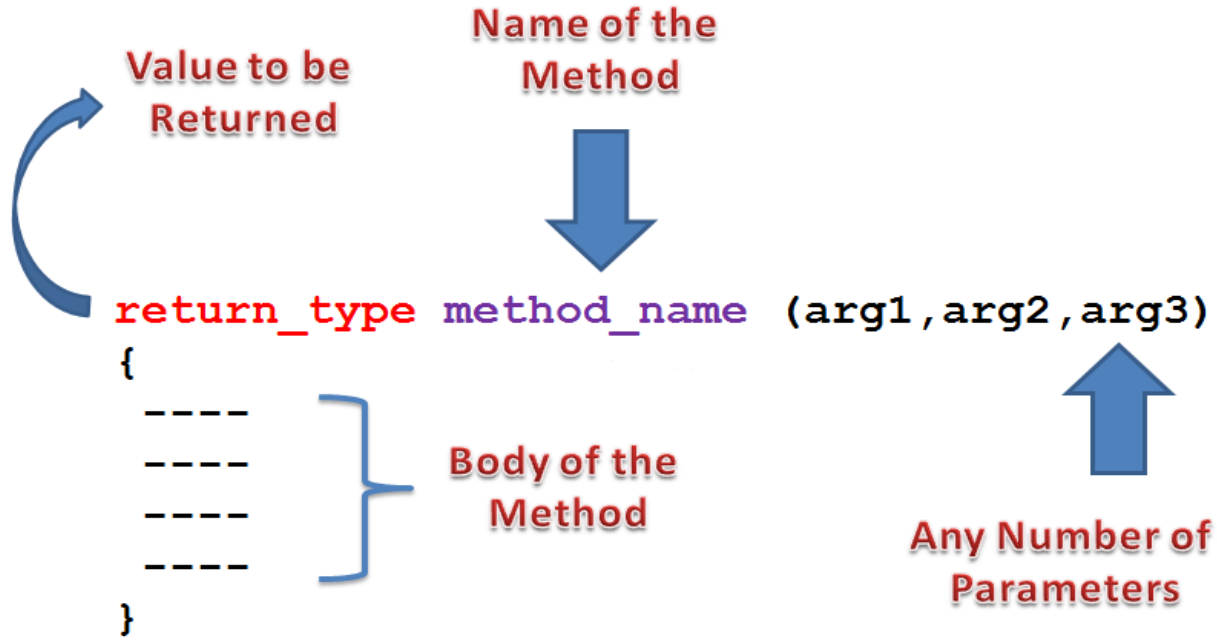
```
public class Student {
    private String name;

    public String getName() {
        return this.name;
    }
     public void setName(String name) {
        this.name = name;
    }
}
```

Специфікатори доступу

	Private	Без модифікатора	Protected	Public
Той же клас	Так	Так	Так	Так
Підклас класу в цьому ж пакеті	Ні	Так	Так	Так
Клас цього ж пакету, що не є підкласом	Ні	Так	Так	Так
Підклас класу в іншому пакеті	Ні	Ні	Так	Так
Клас в другому пакеті, що не є підкласом класу даного пакету	Ні	Ні	Ні	Так

Methods

Methods are ***functions*** that are executed in ***context of object***
Always have ***full access*** to data of object
Method definition consists of a ***method header*** and a ***method body***

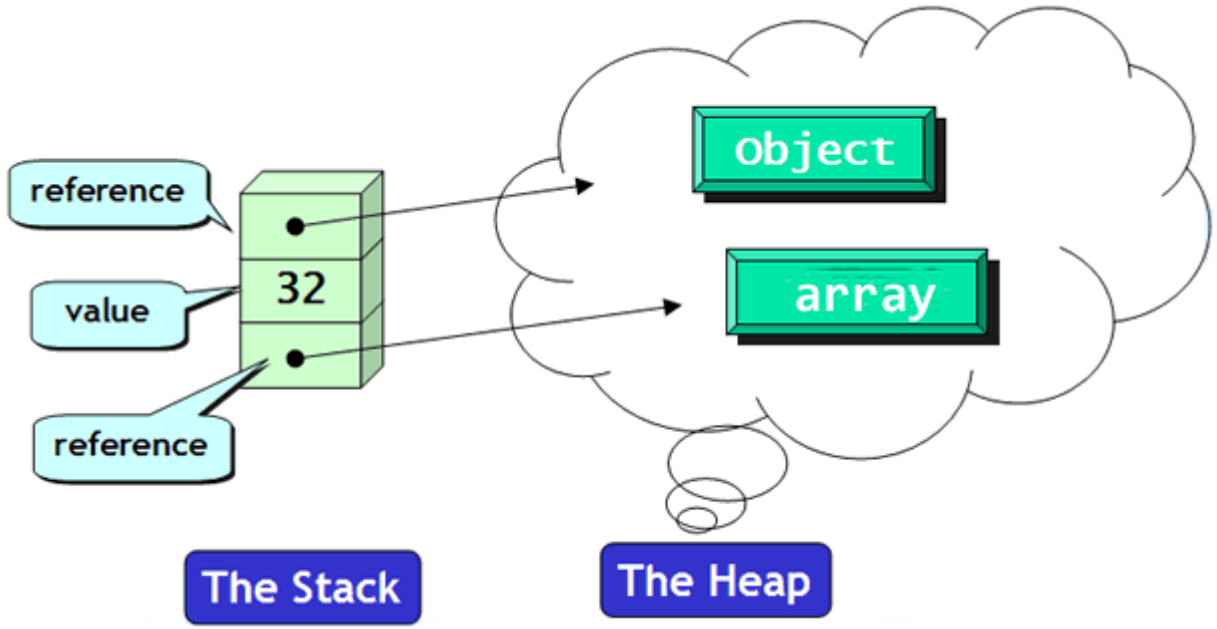


Passing Arguments to Method

- All object references in Java are passed ***by value***
- This means that a ***copy of the value*** will be passed to a method
- Unfortunately, when we pass the ***object***, we are passing the ***copy of reference to it***

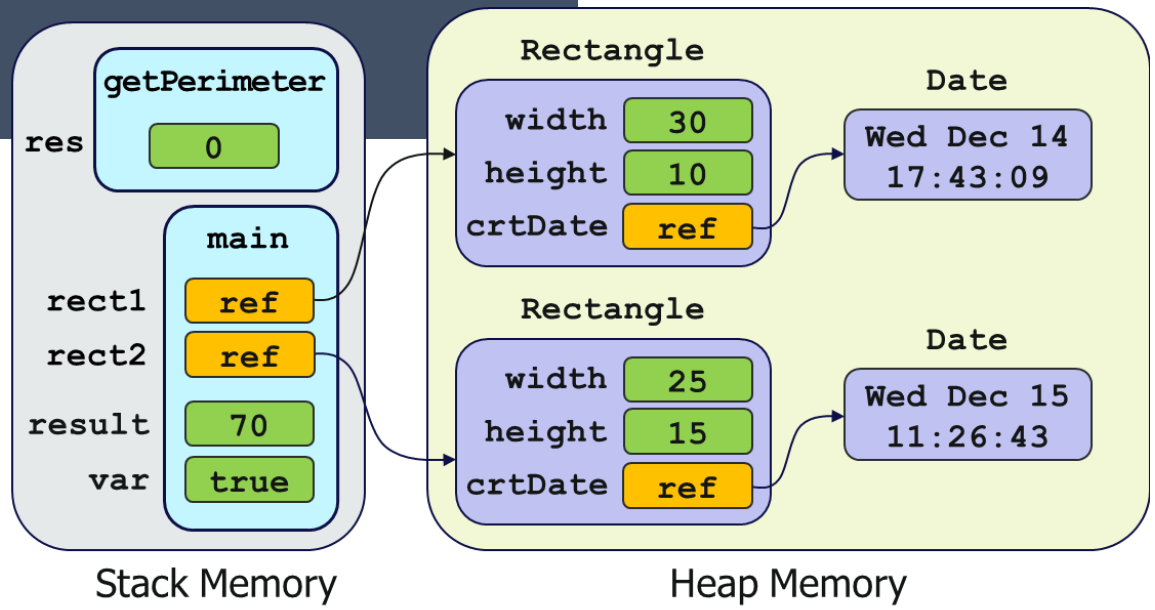
Reference Types

- A **Reference** is a data element that ***holds the address of a memory location***.
- A **Reference Variable** contains a "**pointer**" to an object.

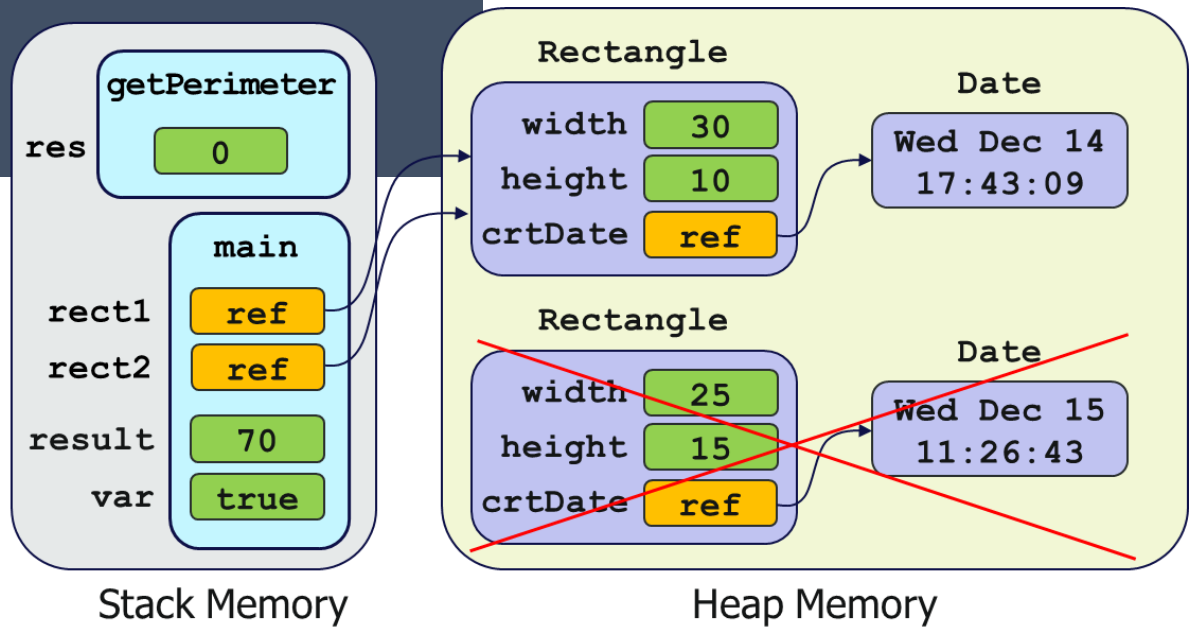


Objects and Assignment Operator

```
public class Main {
    public static void main(String[] args) {
        Rectangle rect1 = new Rectangle(30, 10);
        Rectangle rect2 = new Rectangle(25, 15);

        double result = rect1.getPerimeter();
        boolean var = true;
    }
}
```



```java
public class Main {
    public static void main(String[] args) {
        Rectangle rect1 = new Rectangle(30, 10);
        Rectangle rect2 = rect1;

        double result = rect1.getPerimeter();
        boolean var = true;
    }
}
```



Methods and overloading

- Methods are functions that are executed in context of object
- Always have full access to data of object
- Object can have multiple methods with same name but different signature (type and order of parameters)
- Signature doesn't include return type; methods can't be overloaded by return types

```
class Person {
    String name;
    public void print() {
        System.out.println(name);
    }
    public void print(String s) {
        System.out.println(s + " " + name);
    }
}
```

toString() method overriding

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

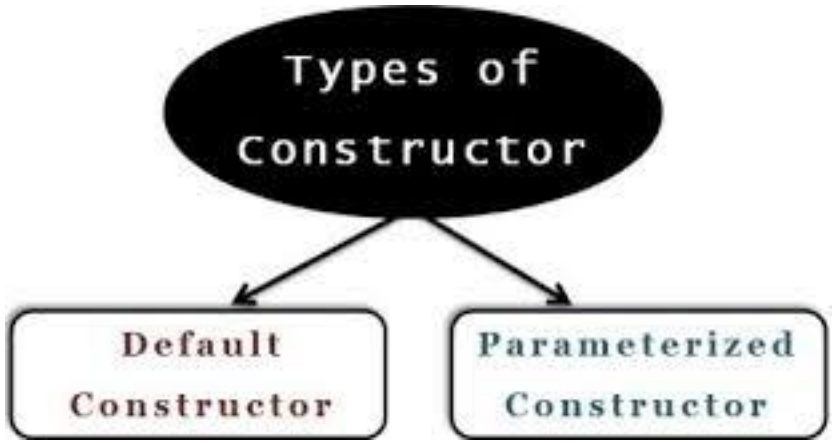
```
com.edu.Student@659e0bfd
```

```
@Override
public String toString() {
      return "Student
       [lastNname=" + lastNname +
         ", firstName=" + firstName +
         ", age=" + age + "]";
}
```

```
Student [lastNname=Ivanov, firstName=Vasiv, age=22]
```

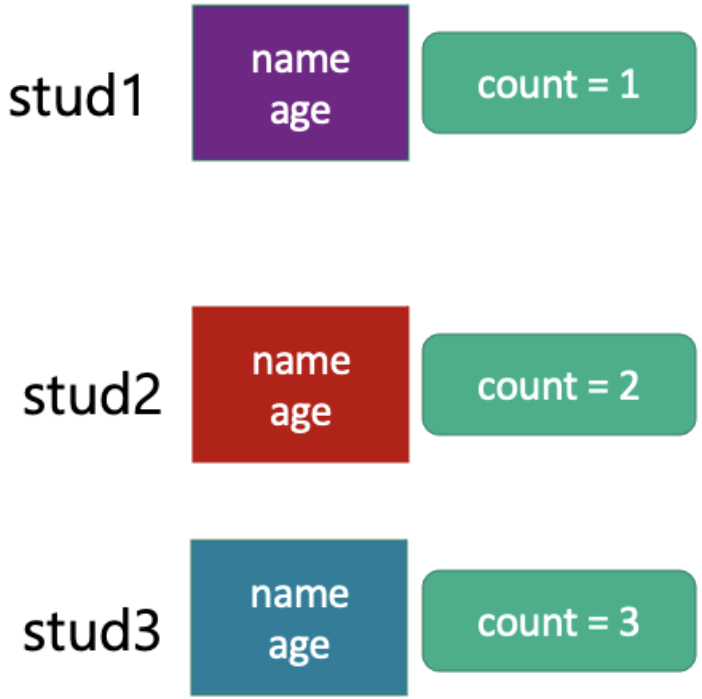
Constructors

- Constructors – special kind of methods called when instance created
- Name should be same as a class
- Class may have multiple overloaded constructors
- If not provided any constructor, Java provides default parameter less empty constructor



Keyword **new.** Creating objects

```
Student stud1 = new Student();
  stud1.setName("Dmytro");
  stud1.setAge(25);

Student stud2 =
      new Student("Olga");
 stud2.setAge(24);

Student stud3 =
      new Student("Ivan", 26);

int n = Student.count;
```



Constructors

```
public class Student {
    private String name;
    private int age;

    public static int count = 0;

    public Student(){count++;}

    public Student(String name){
        this.name = name;
        count++;
    }

    public Student(String name, int age){
        this.name = name;
        this.age = age;
        count++;
    }  ... getters, setters and methods
}
```

They have the same name

Keyword '**this**'

this always points to current object

- not required in most cases
- often needed to distinguish between parameters and fields:

```
public class Person {
    private String name;

    public void setName(String name) {
        this.name = name;
        // name = name;
    }

    public void print() {
        System.out.println(name);
    }
}
```

Keyword '**static**'

- Keyword 'static' indicates that some class member (method or field) is not associated with any particular object
- Static members should be accessible by class name (good practice, not required by language itself)
- You can invoke ***static method*** without any instance of the class to which it belongs

```
public class Helper {
    private static String message;
    public static void setMessage(String message) {
        Helper.message = message;
    }

    public static void print() {
        System.out.println(message);
    }
}
```

Keyword '**static**'

```
public class Runner {

    public static void main (String[] args) {

        Helper.setMessage("hello");
        Helper.print();

        // Not recommended:
        Helper helper = new Helper();
        helper.setMessage("new message");
        helper.print();
    }

}
```

Packages

- **Java package** is a mechanism for organizing Java classes into **namespaces**
- Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

  - A package provides a **unique** namespace for the types it contains.
  - Classes in the same package can access each other's package-access members.

```
package java.awt.event;
```

Packaging

java.lang – basic language functionality and fundamental types

java.util – collection data structure classes

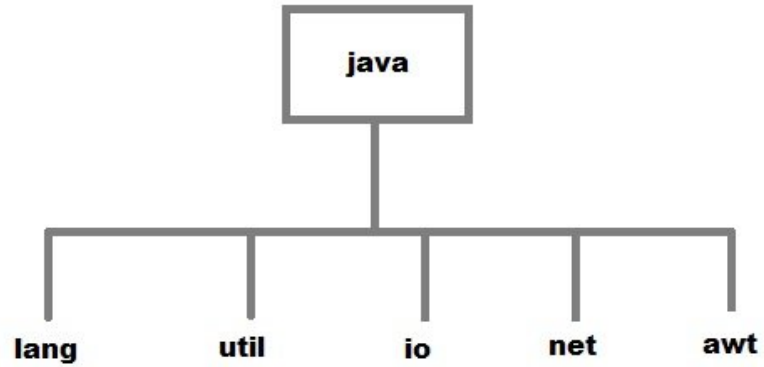
java.io – file operations

java.math – multiprecision arithmetics

java.awt – basic hierarchy of packages for native GUI components

javax.swing – hierarchy of packages for platform-independent rich GUI components

The **java.lang.*** package is available **without** the use of an import statement.



Імпорт пакетів

Імпорт пакетів
Якщо класи знаходяться в іншому пакеті, їх потрібно імпортувати. вказувати повну назву пакету перед використовуваним класом:
java.util.Date today = new java.util.Date();
Такий спосіб незручний, краще використати інструкцію import:
import java.util.Date; // імпортуємо класс
Далі можна використовувати:
Date today = new java.util.Date();

Статичний імпорт

Починаючи з Java SE 5.0 можна зробити так:

```
import static java.lang.System.*;
```

Далі використовуємо:

```
out.println("Goodbye, World!"); // i.e., System.out
exit(0); // i.e., System.exit
```