# Програмування-1

Лекція 5

Основи ООП

# Agenda

- Основні поняття ООП
- Декларування класів
- Створення об’єктів
- Пакети
- Декілька класів, що можуть стати у нагоді ☺

# Основні поняття ООП

- Принципи ООП
  - 0. Абстрагування
  - 1. Інкапсуляція
  - 2. Успадкування
  - 3. Поліморфізм
- Важливі поняття
  - Класи
  - Об’єкти
  - Поля
  - Методи
  - Пакети
  - Область видимості
  - Інтерфейси

# Класи та об’єкти

# Класи та об’єкти

```
class Car {
    String model;
    String color;
    boolean roof;
}
```

```
Car car1 = new Car();
car1.model = "Porsche 911";
car1.color = "green";
car1.roof = true;
```

```
Car car2 = new Car();
car2.model = "Porsche 911";
car2.color = "blue";
car2.roof = false;
```

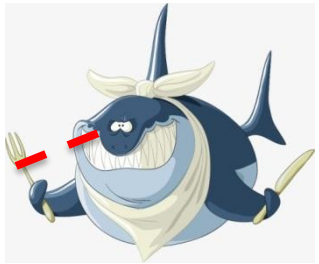
```
Car car3 = new Car();
car3.model = "Porsche 911";
car3.color = "red";
car3.roof = false;
```

# Класи та об’єкти

```
25      public static void main(String[] args) {
26
27          Car car1 = new Car();
28          car1.model = "Porsche 911";
29          car1.color = "green";
30          car1.roof = true;
31
32          Car car2 = new Car();
33          car2.model = car1.model;
34          car2.color = "blue";
35          car2.roof = false;
36
            Car car3 = new Car();
38          car3 = car2;
39          car3.color = "red";
40      }
```

# Класи та об’єкти

```
25  public static void main(String[] args) {
26
27      Car car1 = new Car();
28      car1.model = "Porsche 911";
29      car1.color = "green";
30      car1.roof = true;
31
32      Car car2 = new Car();
33      car2.model = car1.model;
34      car2.color = "blue";
35      car2.roof = false;
36
        Car car3 = new Car();
38      car3 = car2;
39      car3.color = "red";
40  }
```



Garbage collector

# Нагадую

- Об’єкти в Java завжди зберігаються у **Heap**
- Копіювання посилання на об’єкт **не призводить** до копіювання об’єкта

  – копіювання посилання призводить лише до того, що тепер на той самий об’єкт у нас буде ще одне посилання ☺

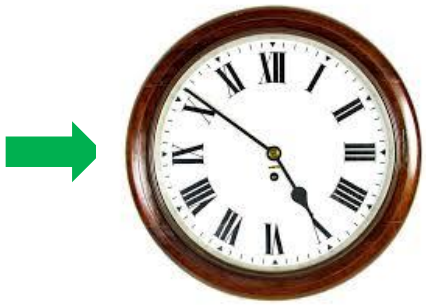
# Абстрагування

- Абстрагування —спосіб виділити набір значущих характеристик об’єкта та виключити незначні

```
public class MyClock {
    public int hours;
    public int minutes;
    public void nextMinute() {
        minutes++;
        if (minutes >=60) {
            minutes=0;
            hours++;
            if (hours >=24) {
                hours=0;
            }
        }
    }
}
```

# Інкапсуляція

- Інкапсуляція — об’єднання даних та методів для роботи з ними
  - публічний інтерфейс + приховання реалізації
    - дані приховують шляхом звуження області видимості
  - дозволяє
    - в майбутньому змінювати реалізацію без зміни інтерфейсу
    - «захист від дурня»



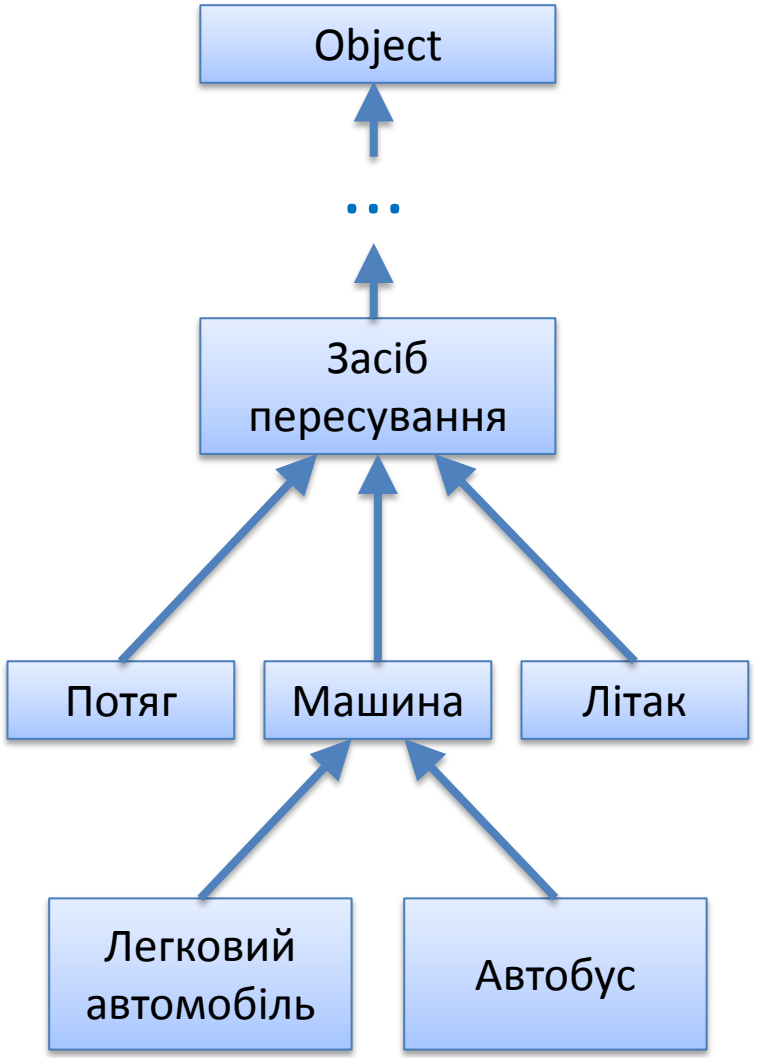
# Інкапсуляція — приклад

```
public class MyClock {
    public int hours;
    public int minutes;

    // ...

}
```

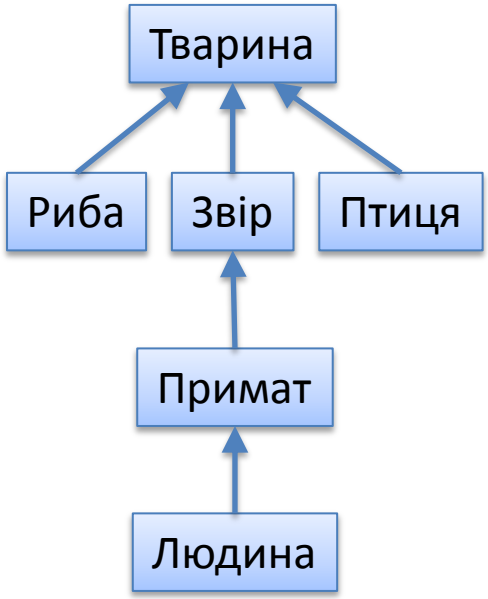
```
public class MyClock {
    private int hours;
    private int minutes;

    public int getHours() {
        return hours;
    }

    public int getMinutes() {
        return minutes;
    }

    // ...

}
```

# Успадкування

- Успадкування — опис нового класу на основі вже існуючого
  - властивості та функціональність батьківського класу переходять до нащадка
- В Java успадкування одиничне
  - у класу є лише один безпосередній предок
  - граф наслідування – дерево
  - відсутня проблема ромбу
  - усі об’єкти прямо або опосередковано успадковані від класу Object



# Успадкування




- За допомогою успадкування реалізують відношення **IS-A** (щось є чимось)
  - Не плутати з **HAS-A** (щось містить щось інше)
  - Це відношення має направлення
    - Будь яка людина є приматом (Людина **extends** Примат)
    - Але не кожен примат є людиною
  - Можна представити у вигляді дерева
    - Людина є Приматом, Примат є Звіром, Звір є Твариною, Птиця є Твариною, Риба є Твариною, Людина не є Рибою
  - Відношення IS-A також працює через кілька рівнів
    - Людина є Звіром, Людина є Твариною

- Для перевірки чи є об’єкт реалізацією певного класу використовують оператор instanceof
  - Працює як безпосередньо так і опосередковано (через кілька рівнів)

```
Людина іванов = new Людина(); // Не використовуйте кирилицю для ідентифікаторів :)
```

    - іванов **instanceof** Людина == **true**
    - іванов **instanceof** Примат == **true**
    - іванов **instanceof** Риба == **false**

# Успадкування— приклад

```
public class MyPreciseClock extends MyClock {
    private int seconds;

    public int getSeconds() {
        return seconds;
    }

    public void nextSecond() {
        seconds++;
        if (seconds>=60) {
                seconds = 0;
                nextMinute();
        }
    }
}

...
```

```
MyClock mc = new MyClock();
MyPreciseClock mpc = new MyPreciseClock();
System.out.println(mc instanceof MyClock);
System.out.println(mc instanceof MyPreciseClock);
System.out.println(mpc instanceof MyClock);
System.out.println(mpc instanceof MyPreciseClock);
```

**Результат:**

```
true
false
true
true
```

# Поліморфізм

- **Поліморфізм** — використання об’єктів з однаковими інтерфейсами без інформації про їх конкретний тип
- Для цього потрібно у підкласах (клас-нащадок) перевизначити методи (override) суперкласу (батьківський клас) або інтерфейсу
  - не плутати **override** та **overload**!!!
    - про overload далі буде :)
- Вибір потрібного метода здійснюється на етапі виконання програми (runtime) на основі інформації про тип об’єкта
    - поліморфізм працює лише з методами!
    - ніколи не перевизначайте поля!!
    - вибір поля здійснюється на етапі компіляції на основі того, до якого типу належить посилання на об’єкт, а не сам об’єкт
- Починаючи з Java 5 рекомендується методи, що перевизначаються, помічати анотацією @Override

# Поліморфізм — приклад 1

```
public class MyClock {
    // ...
    public String toString() {
        return "MyClock [" + hours + ":" + minutes + "]";
    }
}

public class MyPreciseClock extends MyClock {
    // ...
    public String toString() {
        return "MyPreciseClock [" + getHours() + ":" + getMinutes() + ":"
            + seconds + "]";
    }
}


...

MyClock mc = new MyClock();
MyPreciseClock mpc = new MyPreciseClock();
MyClock mpc2 = new MyPreciseClock();

System.out.println(mc);              // MyClock [0:0]
System.out.println(mpc);             // MyPreciseClock [0:0:0]
System.out.println(mpc2);            // MyPreciseClock [0:0:0]
```

# Поліморфізм — приклад 2

```
class A{
   String name = "Class A";
   String getName() {
      return name;
   }
}

class B extends A{
   String name = "Class B"; //!!! Ніколи так не робіть !!!!!
   String getName() {
      return name;
   }
}

public class AB{
   public static void main(String[] args) {
      A a = new A();
      B b = new B();
      A ab= new B();
      System.out.println("a : " + a.name + " " + a.getName());
      System.out.println("b : " + b.name + " " + b.getName());
      System.out.println("ab: " + ab.name+ " " + ab.getName());
   }
}
```

```
Результат:
a : Class A Class A
b : Class B Class B
ab: Class A Class B
```

# Поліморфізм — приклад 3

```
class A{
   String name = "Class A";
   String getName() {
      return name;
   }
}

class B extends A{
   String name = "Class B"; //!!! Ніколи так не робіть !!!!!
   String getName() {
      return name;
   }
}

public class AB{
   public static void main(String[] args) {
      A a = new A();
      B b = new B();
      A ab= new B();
      System.out.println("a : " + a.name + " " + a.getName());
      System.out.println("b : " + b.name + " " + b.getName());
      System.out.println("ab: " + ab.name+ " " + ab.getName());
   }
}
```

```
Результат:
a : Class A Class A
b : Class B Class B
ab: Class A Class B
```

# @Override – приклад

## try to override Object.toString()

```
16  public class Person {
17      String name;
18      String surname;
19
20      @Override
21      public String toString() {
22          return name + " " + surname;
23      }
24
25  }
26
```

```
16  public class Person {
17      String name;
18      String surname;
19
20
21      public String toString() {
22          return name + " " + surname;
23      }
24
25  }
```

```
16  public class Person {
17      String name;
18      String surname;
19
20      @Override
21      public String tostring() {
22          return name + " " + surname;
23      }
24
25  }
```

```
16  public class Person {
17      String name;
18      String surname;
19
20
21      public String tostring() {
22          return name + " " + surname;
23      }
24
25  }
```