Lesson 12

03.03.2025

```
public class Ex1 {
    static int a = 1111;

    static {
        System.out.println("static");
        a = a-- - --a;
    }

    {
        System.out.println("non static");
        a = a++ + ++a;
    }

    public static void main(String[] args) {
        System.out.println(a);
    }
}
```

```
public class Ex2 {
    public static void main(String[] args)  {
        Integer i1 = 128;
        Integer i2 = 128;
        System.out.println(i1 == i2);

        Integer i3 = 127;
        Integer i4 = 127;
        System.out.println(i3 == i4);
    }
}
```

```
public class Ex3 {
    public static void show() {
        System.out.println("Static method called");
    }

    public static void main(String[] args) {
        Ex3 obj = null;
        obj.show();
    }
}
```

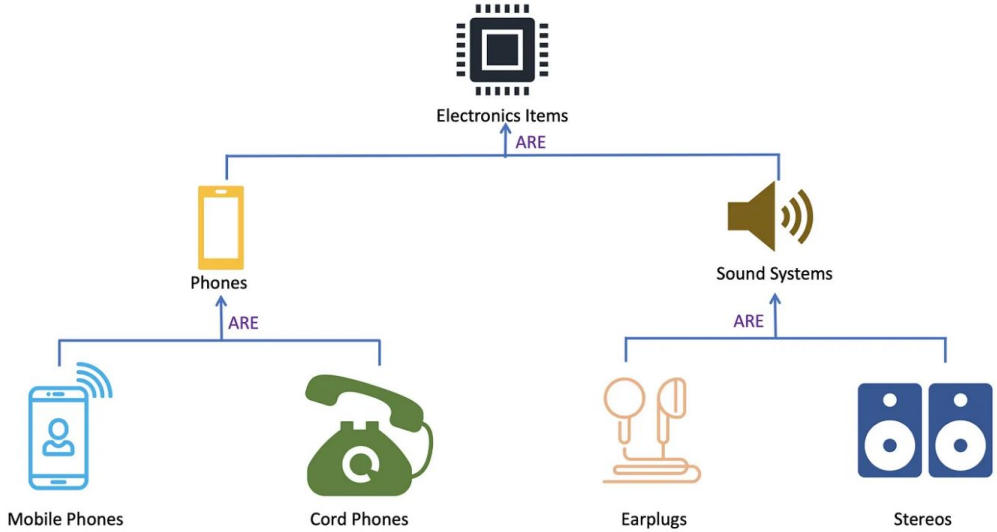
```
public class Ex4 {
    static int method1(int i) {
        return method2(i *= 11);
    }
    static int method2(int i) {
        return method3(i /= 11);
    }
    static int method3(int i) {
        return method4(i -= 11);
    }
    static int method4(int i) {
        return i += 11;
    }
    public static void main(String[] args) {
        System.out.println(method1(11));
    }
}
```

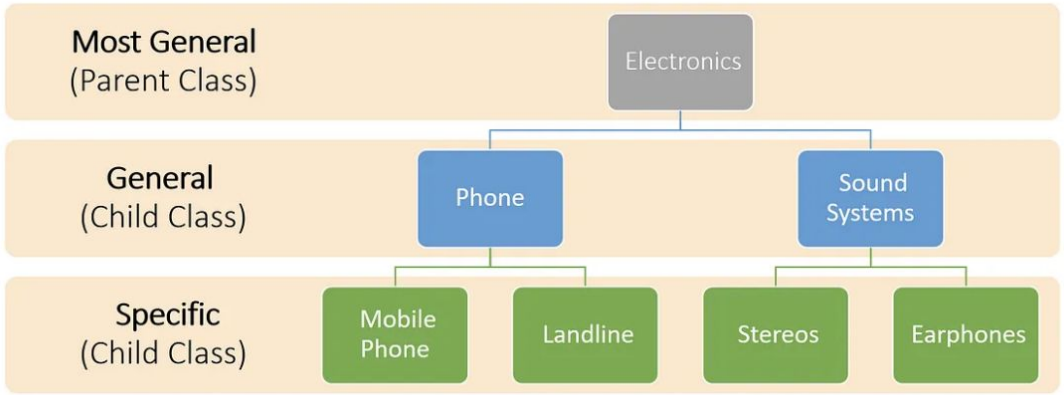
Успадкування

Успадкування — це механізм ООП, який дозволяє створювати нові класи на основі вже існуючих. Новий клас (підклас) отримує всі публічні та захищені поля і методи батьківського класу (суперкласу).

Основні особливості успадкування:

- Дозволяє уникнути дублювання коду.
- Забезпечує повторне використання функціональності.
- Дозволяє створювати ієрархію класів.


Electronics Items
ARE
Phones
ARE
Sound Systems
ARE
Mobile Phones
Cord Phones
Earplugs
Stereos


Most General
(Parent Class)
Electronics
General
(Child Class)
Phone
Sound Systems
Specific
(Child Class)
Mobile Phone
Landline
Stereos
Earphones

Inheritance

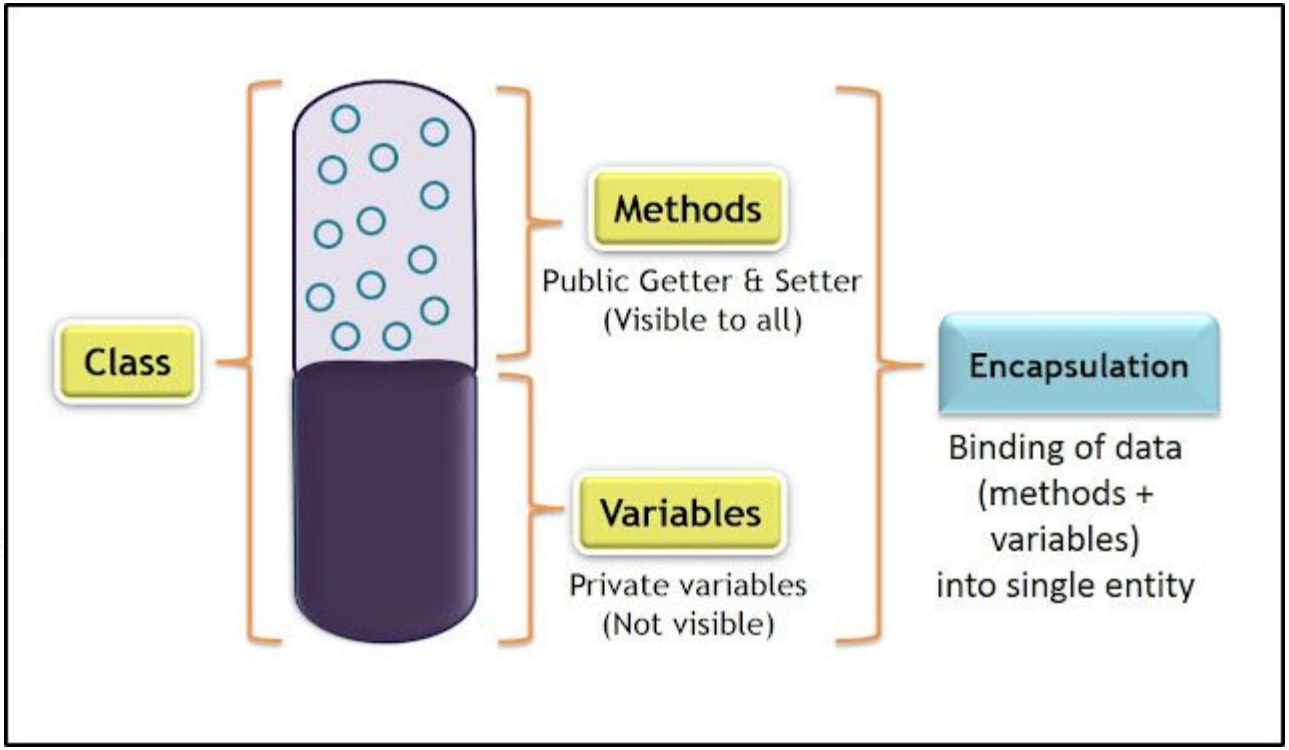
you can create new type of animal
changing or adding properties

Інкапсуляція

Інкапсуляція — це один із ключових принципів об'єктно-орієнтованого програмування (ООП), який полягає у приховуванні деталей реалізації класу та забезпеченні доступу до даних лише через визначені методи.

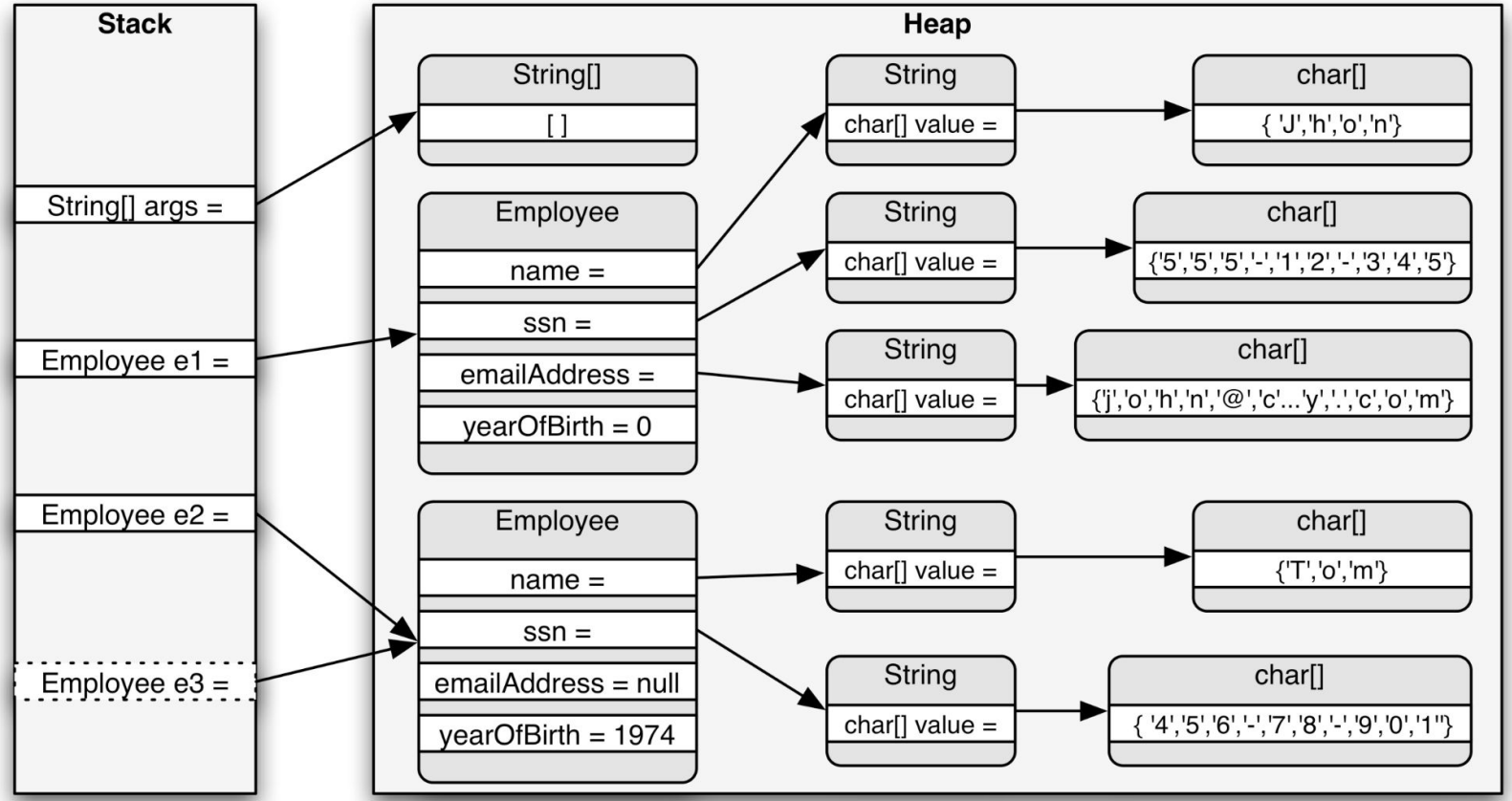
Основні ідеї інкапсуляції:

1. **Приховування реалізації** — внутрішні деталі класу не доступні для зовнішнього використання.
2. **Контроль доступу** — дані можна змінювати лише через спеціально визначені методи.
3. **Захист від некоректних змін** — встановлюючи обмеження доступу, ми запобігаємо неправильному використанню даних.


Class
Methods
Public Getter & Setter
(Visible to all)
Variables
Private variables
(Not visible)
Encapsulation
Binding of data
(methods +
variables)
into single entity

Incapsulation

every animal eats
and then poop


Stack
String[] args =
Employee e1 =
Employee e2 =
Employee e3 =
Heap
String[]
[ ]
Employee
name =
ssn =
emailAddress =
yearOfBirth = 0
Employee
name =
ssn =
emailAddress = null
yearOfBirth = 1974
String
char[] value =
String
char[] value =
String
char[] value =
String
char[] value =
String
char[] value =
char[]
{ 'J','h','o','n'}
char[]
{'5','5','5','-','1','2','-','3','4','5'}
char[]
{'j','o','h','n','@','c'...'y','.','c','o','m'}
char[]
{'T','o','m'}
char[]
{ '4','5','6','-','7','8','-','9','0','1'}

Java variables do not contain the actual objects, they contain *references* to the objects.

- The actual objects are stored in an area of memory known as the *heap*.
- Local variables referencing those objects are stored on the stack.
- More than one variable can hold a reference to the same object.

Клас Object визначає метод clone(), що створює копію об'єкта. Якщо ви хочете, щоб екземпляр вашого класу можна було клонувати, необхідно перевизначити цей метод та реалізувати інтерфейс Cloneable. Інтерфейс Clonable – це інтерфейс-маркер, він не містить ні методів, ні змінних. Інтерфейси-маркер просто визначають поведінку класів.

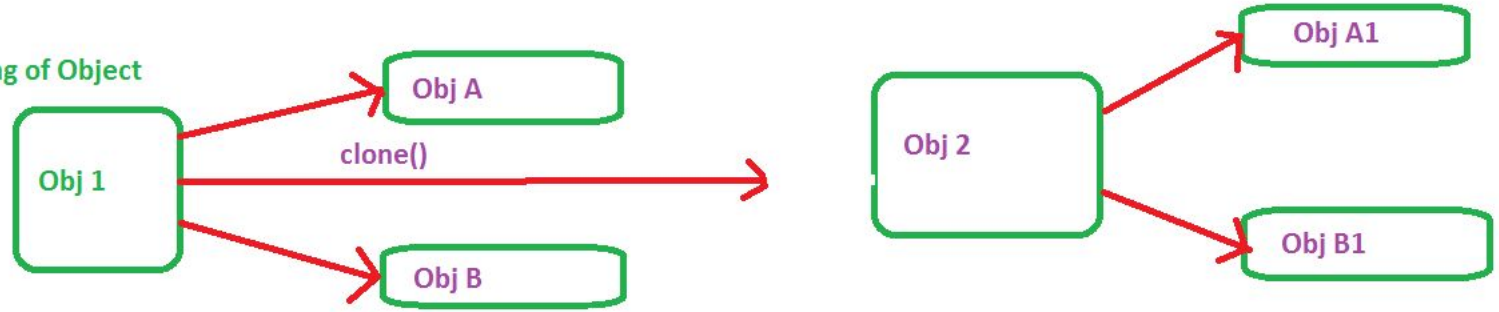
Object.clone() викидає виняток CloneNotSupportedException при спробі клонувати об'єкт, що не реалізує інтерфейс Cloneable. Метод clone() у батьківському класі Object є protected, тому бажано перевизначити його як public. Реалізація за умовчанням методу Object.clone() виконує неповне/поверхневе (shallow) копіювання

```
public class Student implements Cloneable {

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
```

Shallow Cloning of Object

Deep Cloning of Object




this vs super

Чим this і super схожі

- І **this**, і **super** - це нестатичні змінні, відповідно їх не можна використовувати у статичному контексті, а це означає, що їх не можна використовувати у методі main. Це призведе до помилки під час компіляції "на нестатичну змінну цього не можна посилатися зі статичного контексту ". те саме станеться, якщо в методі main скористатися ключовим словом super.
- І **this**, і **super** можуть використовуватися всередині конструкторів для виклику інших конструкторів по ланцюжку, нпр., **this()** і **super()** викликають конструктор без аргументів спадкового та батьківського класів відповідно.
- Всередині конструктора this і super повинні стояти вище за всіх інших виразів, на початку, інакше компілятор видасть повідомлення про помилку. З чого випливає, що в одному конструкторі не може бути одночасно і this(), та super().

Відмінності у super та this

- Змінна this посилається на поточний екземпляр класу, в якому вона використовується, тоді як super - на екземпляр батьківського класу.
- Кожен конструктор за відсутності явних викликів інших конструкторів неявно викликає за допомогою super() конструктор без аргументів батьківського класу, при цьому у вас завжди залишається можливість явно викликати будь-який інший конструктор з допомогою або this(), або super().

this vs super

super

- refer superclass object
- super() can call super class constructor

this

- refer current instance of a class inside the class
- this() can call overloaded constructor