



Lesson 8

28.12.2024

```
public class Ex1 extends Foo {
    public static String sign() {
        return "fa";
    }

    public static void main(String[] args) {
        Ex1 ex = new Ex1();
        Foo foo = new Ex1();

        System.out.println(ex.sign() + " " + foo.sign());
    }
}

class Foo {
    public static String sign() {
        return "la";
    }
}
```

```
public class RedWood extends Tree {
    public static void main(String[] args) {
        new RedWood().go();
    }

    void go() {
        run(new Tree(), new RedWood());
        run((Tree) new RedWood(), (RedWood) new Tree());
    }

    void run(Tree t1, RedWood r1) {
        RedWood r2 = (RedWood) t1;
        Tree t2 = (Tree) r1;
    }
}

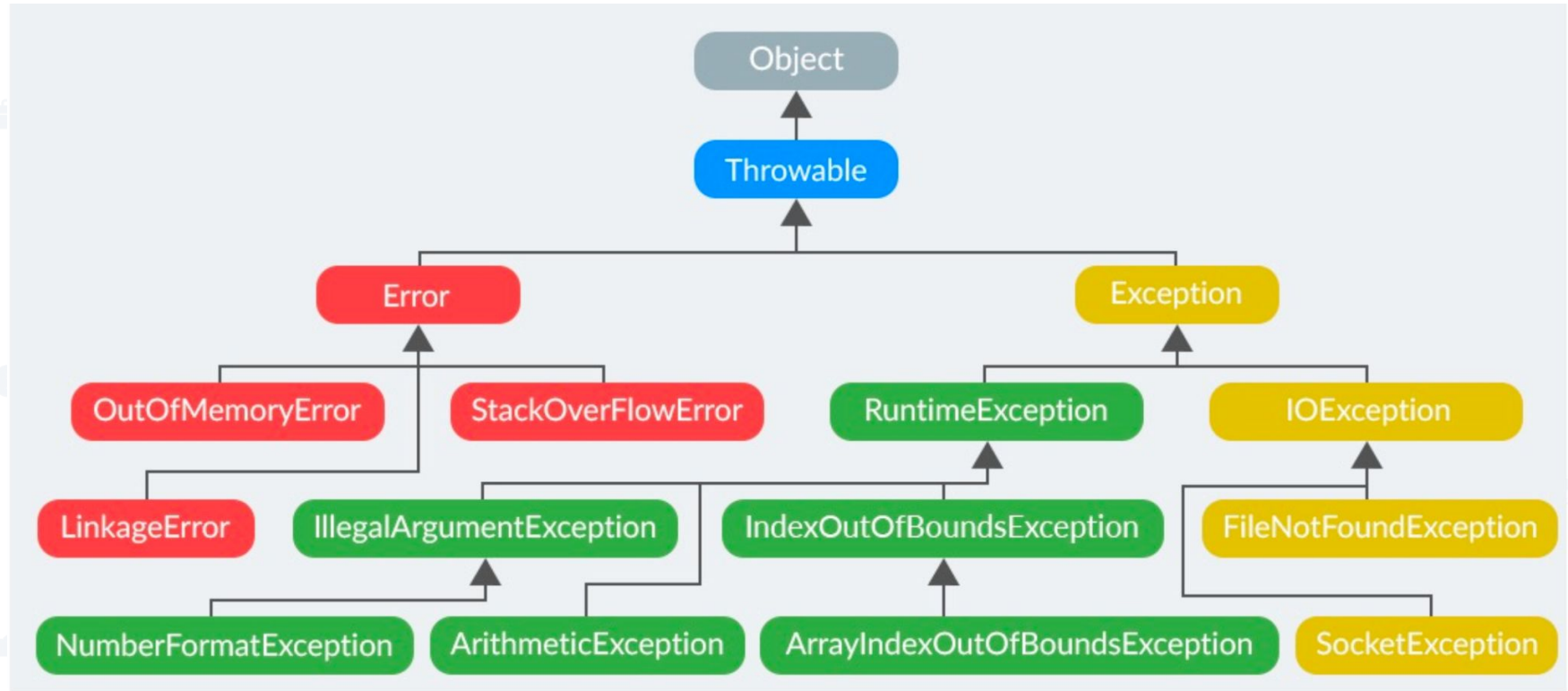
class Tree {
}
```

```
public class Ex3 extends Electronic implements Gadget {  
    public void doSomething() {  
        System.out.println("serf internet ...");  
    }  
  
    public static void main(String[] args) {  
        new Ex3().doSomething();  
        new Ex3().getPower();  
    }  
}  
  
abstract class Electronic{  
    void getPower(){  
        System.out.println("plug in ...");  
    }  
}  
  
interface Gadget{  
    void doSomething();  
}
```

```
public class Ex4 {  
    public static void main(String[] args) {  
        int numFish = 4;  
        String fishType = "tuna";  
        String anotherFish = numFish + 1;  
        System.out.println(anotherFish + " " + fishType);  
        System.out.println(numFish + " " + 1);  
    }  
}
```

```
public class Ex5 {  
    private final static String RESULT ="2cfalse";  
    public static void main(String[] args) {  
        String a = "";  
        a += 2;  
        a += 'c';  
        a += false;  
        if (a == RESULT) System.out.println("==");  
        if (a.equals(RESULT)) System.out.println("equals");  
    }  
}
```

```
public class Ex6 {  
    final static short x = 2;  
    public static int y = 0;  
  
    public static void main(String[] args) {  
        for (int z =0; z < 3; z++){  
            switch (z){  
                case x: System.out.print("0 ");  
                case x - 1: System.out.print("1 ");  
                case x - 2: System.out.print("2 ");  
            }  
        }  
    }  
}
```



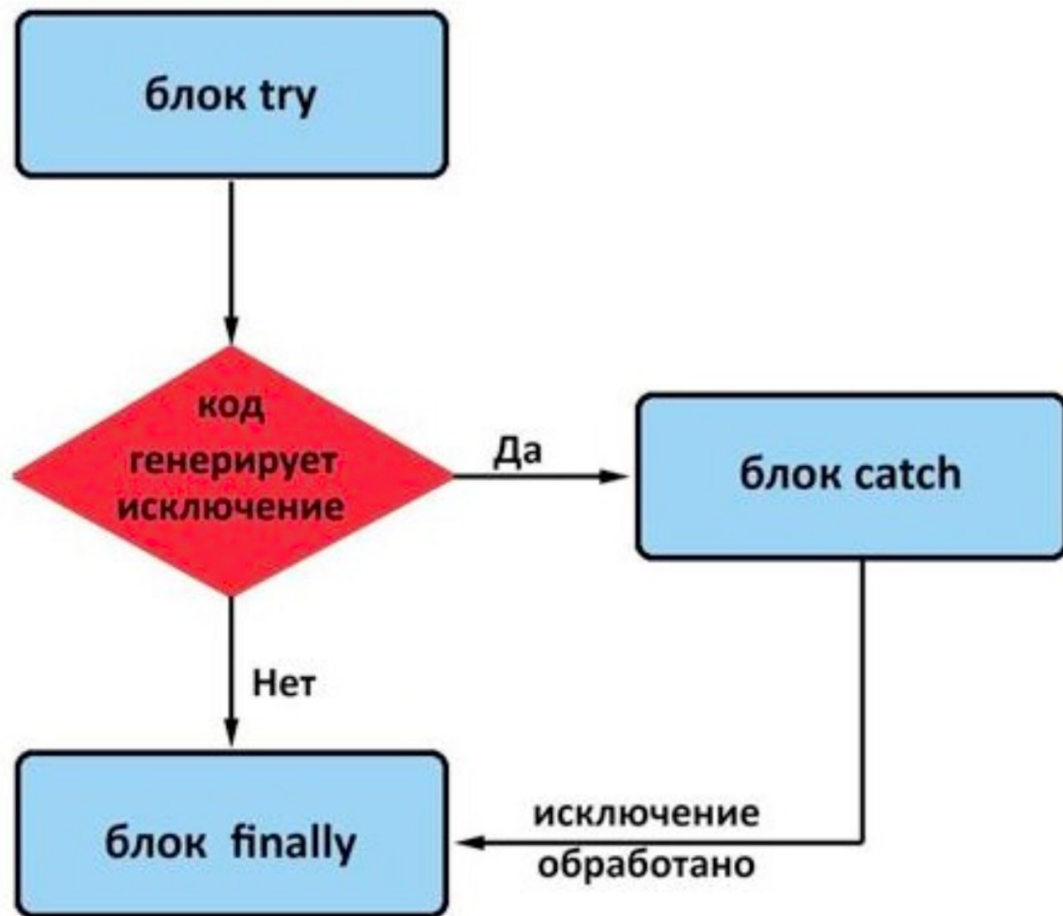
try – визначає блок коду, в якому може бути виключено;

catch – визначає блок коду, в якому відбувається обробка виключення;

finally – визначає код блоку, який не є обов'язковим, але при його наявності виконується в будь-якому випадку незалежно від результатів виконання блоку try.

throw – використовується для возбуждення виключення;

throws – використовується в сигнатурі методів для попередження, про те, що метод може викинути виключення.



```
public class TryCatch {  
    public static void main(String[] args) {  
        try {  
            System.err.print(" 0");  
            if (true) {  
                throw new RuntimeException();  
            }  
            System.err.print(" 1");  
        } catch (RuntimeException e) {  
            System.err.print(" 2");  
        }  
        System.err.println(" 3");  
    }  
}
```

try + catch + catch + ...

```
public class TryCatchCatch {  
    public static void main(String[] args) {  
        try {  
            throw new Exception();  
        } catch (RuntimeException e) {  
            System.err.println("catch RuntimeException");  
        } catch (Exception e) {  
            System.err.println("catch Exception");  
        } catch (Throwable e) {  
            System.err.println("catch Throwable");  
        }  
        System.err.println("next statement");  
    }  
}
```

```
public class TryFinally {  
    public static void main(String[] args) {  
        try {  
            throw new RuntimeException();  
        } finally {  
            System.err.println("finally");  
        }  
    }  
}
```

try + catch + finally


```
public class TryCatchFinally {  
    public static void main(String[] args) {  
        try {  
            System.err.print(" 0");  
            if (true) {throw new Error();}  
            System.err.print(" 1");  
        } catch(Error e) {  
            System.err.print(" 2");  
        } finally {  
            System.err.print(" 3");  
        }  
        System.err.print(" 4");  
    }  
}
```



try-with-resources

```
public static void main(String[] args) {
    Scanner scanner = null;
    try {
        scanner = new Scanner(new File( pathname: "test.txt"));
        while (scanner.hasNext()) {
            System.out.println(scanner.nextLine());
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } finally {
        if (scanner != null) {
            scanner.close();
        }
    }
}

public static void main(String[] args) {
    try (Scanner scanner = new Scanner(new File( pathname: "test.txt"))) {
        while (scanner.hasNext()) {
            System.out.println(scanner.nextLine());
        }
    } catch (FileNotFoundException fnfe) {
        fnfe.printStackTrace();
    }
}
```

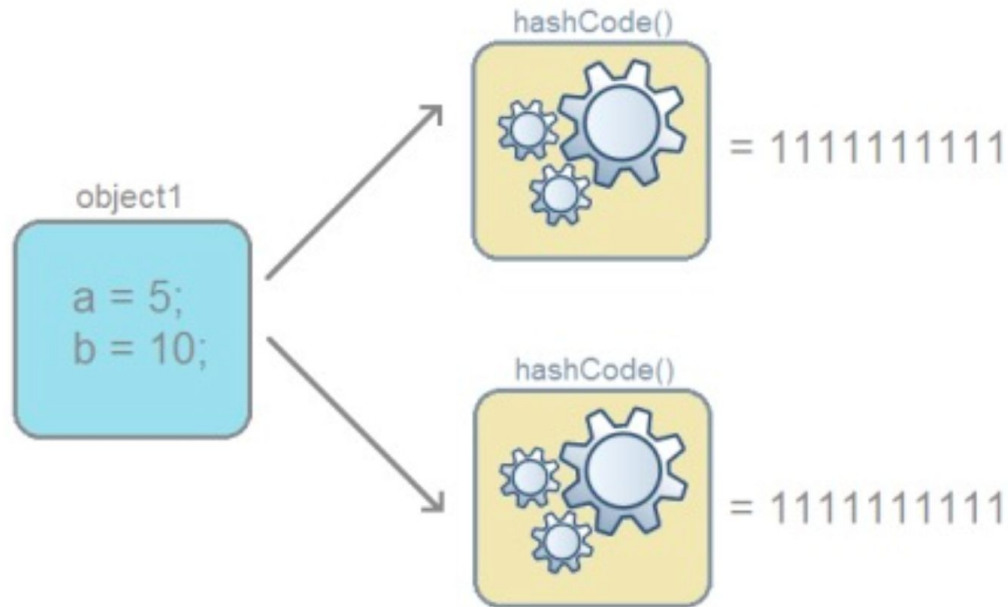


Необхідно розуміти, що

- перевірка виключення **checked** відбувається в момент компіляції (перевірка під час компіляції)
- перехват виключення (catch) відбувається в момент виконання (перевірка виконання)

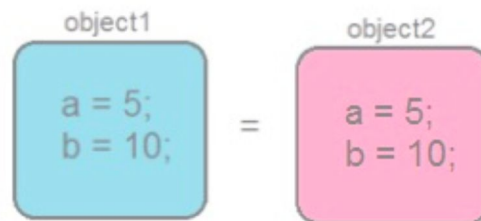
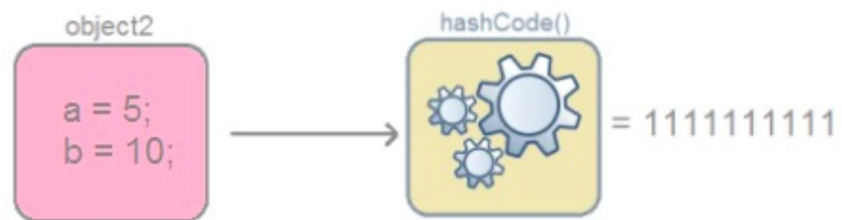
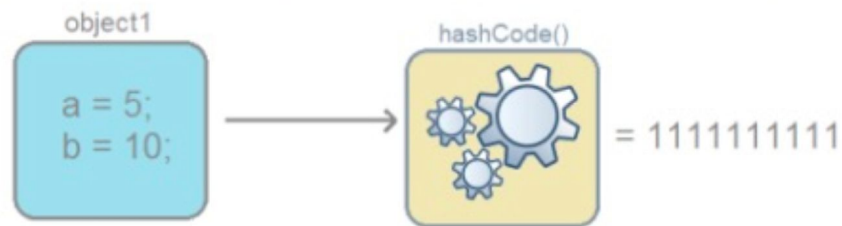
hashCode() та equals()

Для одного і того ж об'єкта, хеш-код завжди буде однаковим



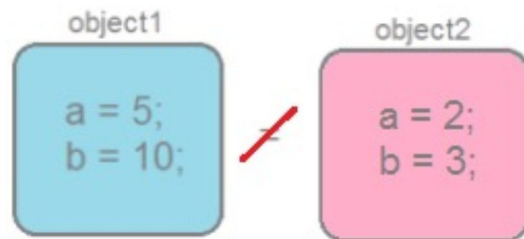
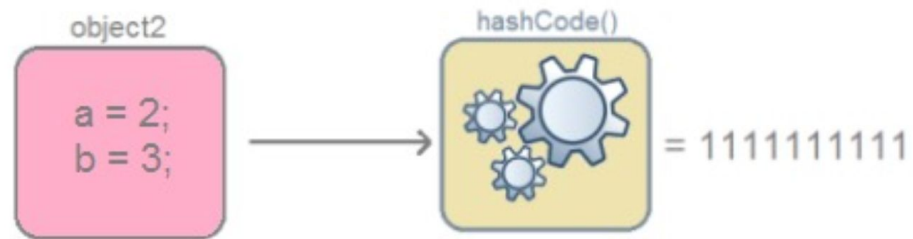
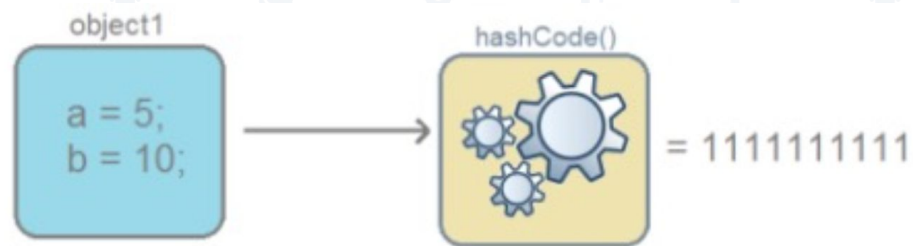
1111111111 = 1111111111

Якщо об'єкти однакові, то і хеш-коди однакові



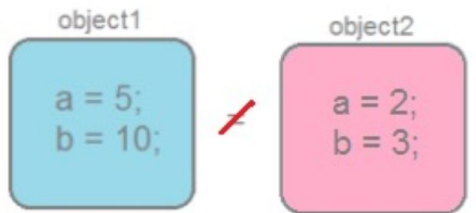
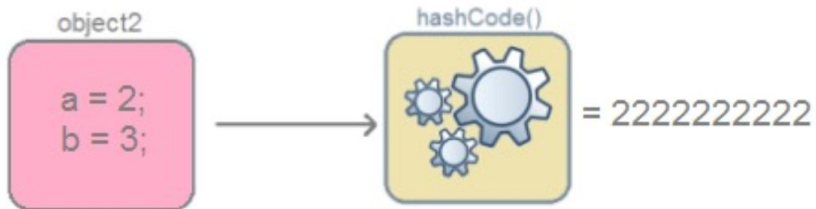
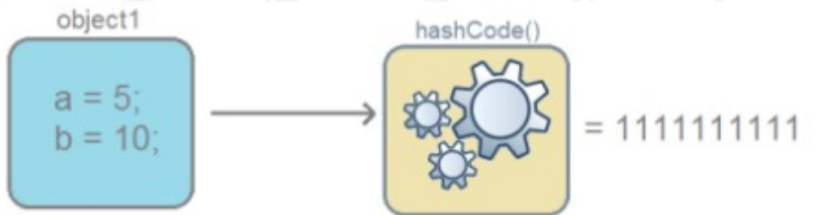
1111111111 = 1111111111

Якщо хеш-коди рівні, то вхідні об'єкти не завжди рівні (колізія)





Якщо хеш-коди різні, то об'єкти гарантовано різні.



1111111111 ~~≠~~ 2222222222



equals() - контракт

1 - Рефлексивність

для будь-якого заданого значення x вираз $x.equals(x)$ повинен повертати `true`. Заданого - мається на увазі такого, що $x \neq null$

2 - Симетричність

для будь-яких заданих значень x та y , $x.equals(y)$ має повертати `true` тільки в тому випадку, коли $y.equals(x)$ повертає `true`.

3 - Транзитивність

для будь-яких заданих значень x , y та z , якщо $x.equals(y)$ повертає `true` і $y.equals(z)$ повертає `true`, $x.equals(z)$ має повернути значення `true`.

4 - Узгодженість

для будь-яких заданих значень x та y повторний виклик $x.equals(y)$ буде повертати значення попереднього виклику цього методу за умови, що поля, які використовуються порівняння цих двох об'єктів, не змінювалися між викликами.

5 - Порівняння null

для будь-якого заданого значення x виклик $x.equals(null)$ має повертати `false`



Коли можна не перевизначати цей метод

1 - Коли кожен екземпляр класу є унікальним. (Enum, Thread)

2 - Коли насправді від класу не потрібно визначати еквівалентність його екземплярів.

Наприклад, для класу `java.util.Random` взагалі немає необхідності порівнювати між собою екземпляри класу, визначаючи, чи можуть вони повернути однакову послідовність випадкових чисел. Просто тому, що природа цього класу навіть не має на увазі таку поведінку.

3 - Коли клас, який ви розширюєте, вже має свою реалізацію методу `equals` та поведінка цієї реалізації вас влаштовує.

4 - Немає необхідності перекривати `equals`, коли область видимості вашого класу є `private` або `package-private` і ви впевнені, що цей метод ніколи не буде викликано



hashCode() - контракт

1 - виклик методу hashCode() один і більше разів над тим самим об'єктом повинен повертати те саме хеш-значення, за умови що поля об'єкта, значення, що беруть участь у обчисленні, не змінювалися.

2 - виклик методу hashCode() над двома об'єктами повинен завжди повертати те саме число, якщо ці об'єкти рівні (виклик методу equals() для цих об'єктів повертає true).

3 - виклик методу hashCode() над двома нерівними між собою об'єктами повинен повертати різні хеш-значення. Хоча ця вимога і не є обов'язковим слід враховувати, що його виконання позитивно вплине на продуктивність роботи хеш-таблиць.



equals

hashCode





Enum

Крім окремих примітивних типів даних та класів у Java є такий тип як enum чи перерахування. Переліки представляють набір логічно пов'язаних констант. Оголошення перерахування відбувається за допомогою оператора enum, після якого йде назва перерахування. Потім іде список елементів перерахування через кому:

```
public enum UserStatus {  
    PENDING,  
    ACTIVE,  
    INACTIVE,  
    DELETED;  
}
```