

Java

Curs 4

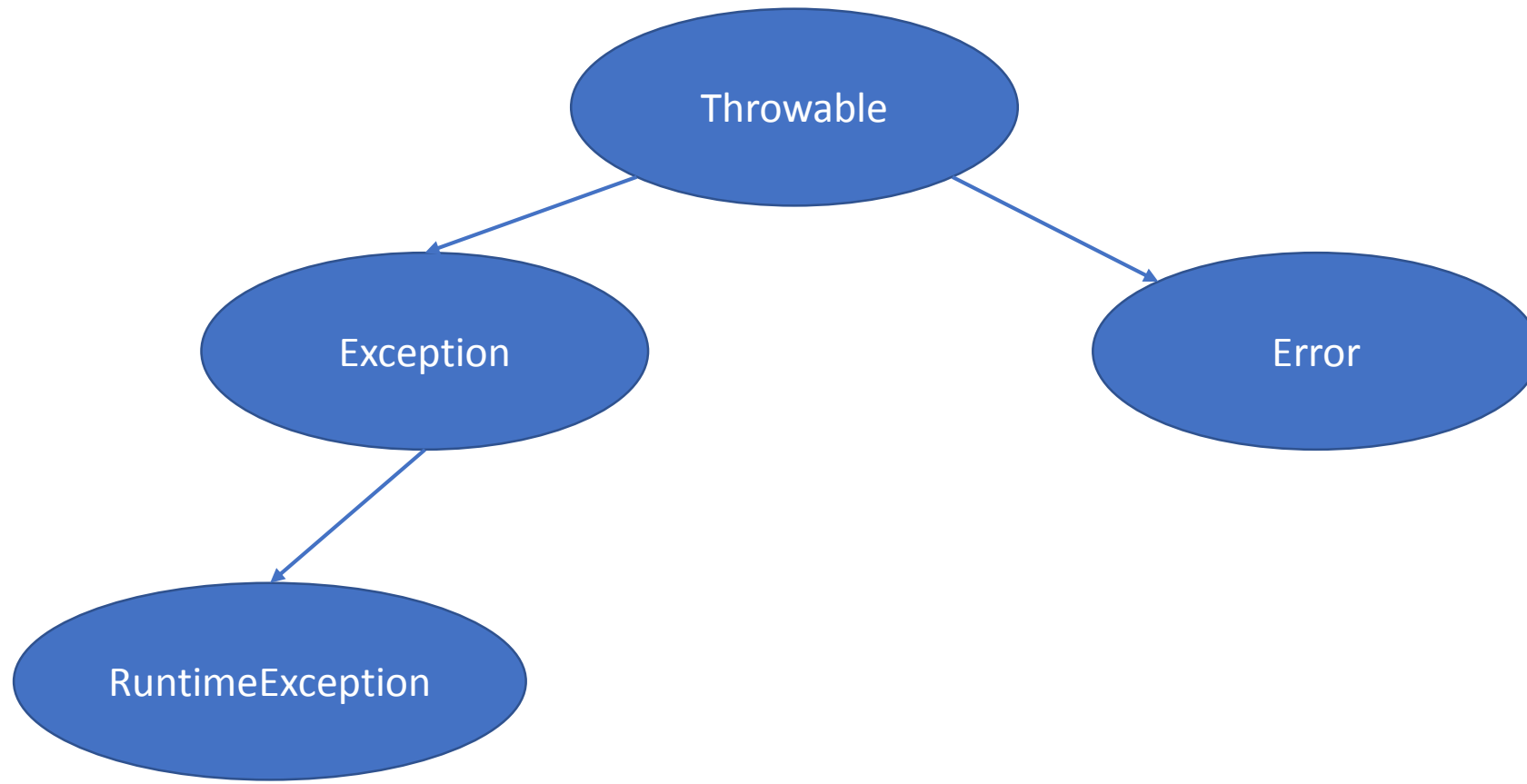
Teme

- Tratarea exceptiilor in Java
- Enumerari
- Autoboxing
- Tipuri generice

Exceptii

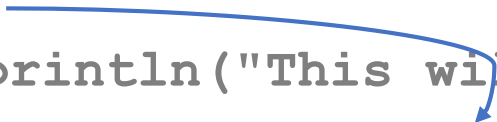
```
try {  
    // block of code to monitor for errors  
}  
catch (ExceptionType1 exOb) {  
    // exception handler for ExceptionType1  
}  
catch (ExceptionType2 exOb) {  
    // exception handler for ExceptionType2  
}  
// ...  
finally {  
    // block of code to be executed after try block ends  
}
```

Ierarhia exceptiilor



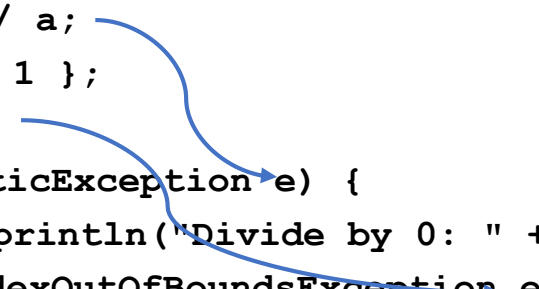
Exemplu

```
class Exc2 {  
    public static void main(String args[]) {  
        int d, a;  
        try { // monitor a block of code.  
            d = 0;  
            a = 42 / d;  
            System.out.println("This will not be printed.");  
        } catch (ArithmeticException e) {  
            // catch divide-by-zero error  
            System.out.println("Division by zero.");  
        }  
        System.out.println("After catch statement.");  
    }  
}
```



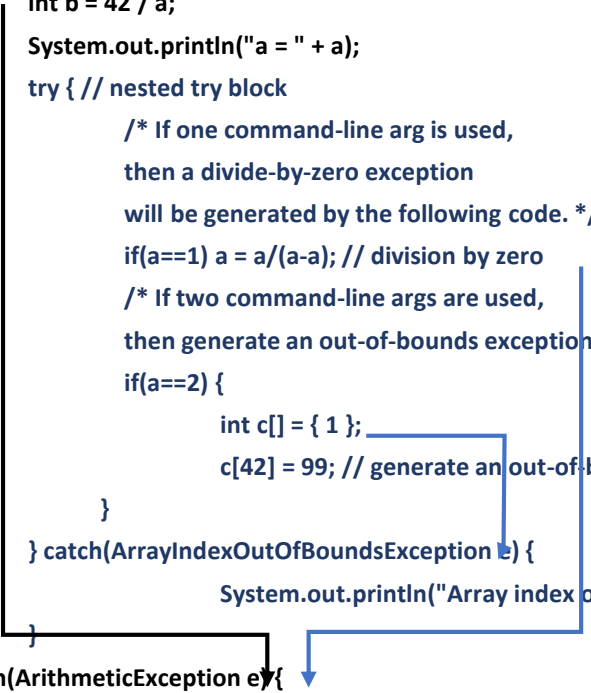
Mai multe catch-uri

```
// Demonstrate multiple catch statements.
class MultipleCatches {
public static void main(String args[]) {
    try {
        int a = args.length;
        System.out.println("a = " + a);
        int b = 42 / a;
        int c[] = { 1 };
        c[42] = 99;
    } catch (ArithmeticException e) {
        System.out.println("Divide by 0: " + e);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks.");
}
}
```

A blue line diagram illustrating the flow of exception handling. It starts from the line `c[42] = 99;` in the `try` block, curves down and to the right, pointing to the `ArrayIndexOutOfBoundsException` catch block. From there, it curves down and to the left, pointing to the `ArithmeticException` catch block. This indicates that the `ArrayIndexOutOfBoundsException` is caught first, and then the `ArithmeticException` is caught.

Exceptii imbricate

```
class NestTry {  
    public static void main(String args[]) {  
        try {  
            int a = args.length;  
            /* If no command-line args are present,  
            the following statement will generate  
            a divide-by-zero exception. */  
            int b = 42 / a;  
            System.out.println("a = " + a);  
            try { // nested try block  
                /* If one command-line arg is used,  
                then a divide-by-zero exception  
                will be generated by the following code. */  
                if(a==1) a = a/(a-a); // division by zero  
                /* If two command-line args are used,  
                then generate an out-of-bounds exception. */  
                if(a==2) {  
                    int c[] = { 1 };  
                    c[42] = 99; // generate an out-of-bounds exception  
                }  
            } catch (ArrayIndexOutOfBoundsException e) {  
                System.out.println("Array index out-of-bounds: " + e);  
            }  
        } catch (ArithmeticException e) {  
            System.out.println("Divide by 0: " + e);  
        }  
    }  
}
```



throw

```
// Demonstrate throw.

class ThrowDemo {
    static void demoproc() {
        try {
            throw new NullPointerException("demo");
        } catch(NullPointerException e) {
            System.out.println("Caught inside demoproc.");
            throw e; // rethrow the exception
        }
    }
}

public static void main(String args[]) {
    try {
        demoproc();
    } catch(NullPointerException e) {
        System.out.println("Recaught: " + e);
    }
}
}
```



throws

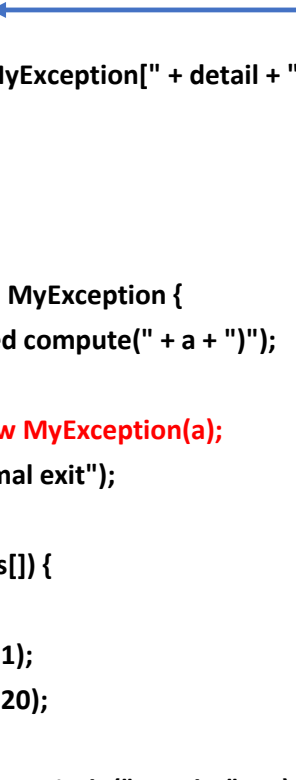
```
class ThrowsDemo {  
    static void throwOne() throws IllegalAccessException {  
        System.out.println("Inside throwOne.");  
        throw new IllegalAccessException("demo");  
    }  
    public static void main(String args[]) {  
        try {  
            throwOne();  
        } catch (IllegalAccessException e) {  
            System.out.println("Caught " + e);  
        }  
    }  
}
```

Exemple de exceptii

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Exceptii custom

```
class MyException extends Exception {  
    private int detail;  
    MyException(int a) {  
        detail = a;  
    }  
    public String toString() {  
        return "MyException[" + detail + "]";  
    }  
}  
  
class ExceptionDemo {  
    static void compute(int a) throws MyException {  
        System.out.println("Called compute(" + a + ")");  
        if(a > 10)  
            throw new MyException(a);  
        System.out.println("Normal exit");  
    }  
    public static void main(String args[]) {  
        try {  
            compute(1);  
            compute(20);  
        } catch (MyException e) {  
            System.out.println("Caught " + e);  
        }  
    }  
}
```



Rezultat:

Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]

Exceptii in lantuite

```
class MultiCatch {  
    public static void main(String args[]) {  
        int a=10, b=0;  
        int vals[] = { 1, 2, 3 };  
        try {  
            int result = a / b; // generate an ArithmeticException  
            // vals[10] = 19; // generate an ArrayIndexOutOfBoundsException  
            // This catch clause catches both exceptions.  
        }  
        catch(ArithmeticException | ArrayIndexOutOfBoundsException e) {  
            System.out.println("Exception caught: " + e);  
        }  
        System.out.println("After multi-catch.");  
    }  
}
```

Enumerari

```
// An enumeration of apple varieties.
```

```
enum Apple {  
    Jonathan, GoldenDel, RedDel, Winesap, Cortland  
}
```

```
class EnumDemo {  
public static void main(String args[])  
{  
    Apple ap;  
    ap = Apple.RedDel;  
    // Output an enum value.  
    System.out.println("Value of ap: " + ap);  
    System.out.println();  
    ap = Apple.GoldenDel;  
    // Compare two enum values.  
    if(ap == Apple.GoldenDel)  
        System.out.println("ap contains GoldenDel.\n");  
    // Use an enum to control a switch statement.  
    switch(ap) {  
        case Jonathan:  
            System.out.println("Jonathan is red.");  
            break;  
        case GoldenDel:  
            System.out.println("Golden Delicious is yellow.");  
            break;
```

```
        case RedDel:  
            System.out.println("Red Delicious is red.");  
            break;  
        case Winesap:  
            System.out.println("Winesap is red.");  
            break;  
        case Cortland:  
            System.out.println("Cortland is red.");  
            break;
```

```
}
```

```
}
```

```
}
```

Values si valueOf

```
class EnumDemo2 {  
    public static void main(String args[])  
    {  
        Apple ap;  
        System.out.println("Here are all Apple constants:");  
        // use values()  
        Apple allapples[] = Apple.values();  
        for(Apple a : allapples)  
            System.out.println(a);  
        System.out.println();  
        // use valueOf()  
        ap = Apple.valueOf("Winesap");  
        System.out.println("ap contains " + ap);  
    }  
}
```

Enumerarea este o clasa extinsa din `java.lang.Enum`

```
enum Apple {  
    Jonathan(10), GoldenDel(9), RedDel(12),  
    Winesap(15), Cortland(8);  
    private int price; // price of each apple  
    // Constructor  
    Apple(int p) { price = p; }  
    int getPrice() { return price; }  
}  
  
class EnumDemo3 {  
    public static void main(String args[])  
    {  
        Apple ap;  
        // Display price of Winesap.
```

```
        System.out.println("Winesap costs " +  
        Apple.Winesap.getPrice() +  
        " cents.\n");  
        // Display all apples and prices.  
        System.out.println("All apple prices:");  
        for(Apple a : Apple.values())  
            System.out.println(a + " costs " +  
            a.getPrice() + " cents.");  
    }  
}
```

Autoboxing

```
class AutoBox {  
    // Take an Integer parameter and return  
    // an int value;  
    static int m(Integer v) {  
        return v ; // auto-unbox to int  
    }  
    public static void main(String args[]) {  
        // Pass an int to m() and assign the return value  
        // to an Integer. Here, the argument 100 is autoboxed  
        // into an Integer. The return value is also autoboxed  
        // into an Integer.  
        Integer iOb1 = 100; // autobox an int  
        int i = iOb1; // auto-unbox  
        Integer iOb = m(100);  
        System.out.println(iOb);  
    }  
}
```

AutoBoxing

```
class AutoBox {  
    public static void main(String args[]) {  
        // Autobox/unbox a boolean.  
        Boolean b = true;  
        // Below, b is auto-unboxed when used in  
        // a conditional expression, such as an if.  
        if(b) System.out.println("b is true");  
        // Autobox/unbox a char.  
        Character ch = 'x'; // box a char  
        char ch2 = ch; // unbox a char  
        System.out.println("ch2 is " + ch2);  
    }  
}
```

Generice

```
class Gen<T> {  
    T ob; // declare an object of type T  
    // Pass the constructor a reference to  
    // an object of type T.  
    Gen(T o) {  
        ob = o;  
    }  
    // Return ob.  
    T getob() {  
        return ob;  
    }  
    void showType() {  
        System.out.println("Type of T is " +  
            ob.getClass().getName());  
    }  
}
```

Type Variable Name	Meaning
E	Element type in a collection
K	Key type in a map
V	Value type in a map
T	General type
S, U	Additional general types

Forma generala:

```
accessSpecifier class GenericClassName<TypeVariable1,  
TypeVariable2, . . . >  
{  
    constructors  
    methods  
    fields  
}
```

Generic - exemplu

```
// Demonstrate the generic class.
class GenDemo {
public static void main(String args[]) {
    // Create a Gen reference for
    // Integers.
    Gen<Integer> iOb;
    // Create a Gen<Integer> object and
    // assign its
    // reference to iOb. Notice the use of
    // autoboxing
    // to encapsulate the value 88 within
    // an Integer object.
    iOb = new Gen<Integer>(88);
    // Show the type of data used by iOb.
    iOb.showType();
    // Get the value in iOb. Notice that
    // no cast is needed.
}
```

```
int v = iOb.getob();
System.out.println("value: " + v);
System.out.println();
// Create a Gen object for Strings.
Gen<String> strOb = new Gen<String>
("Generics Test");
// Show the type of data used by
strOb.
strOb.showType();
// Get the value of strOb. Again,
notice
// that no cast is needed.
String str = strOb.getob();
System.out.println("value: " + str);
}
```

Cateva reguli

- Genericele functioneaza doar cu referinte
 - `Gen<int> intOb = new Gen<int>(53);` // Error, can't use primitive type
- Tipurile de baza a datelor pot fi diferite conform argumentelor
 - `iOb = strOb;` // Wrong!

De ce generice si nu Object?

```
class NonGen {
    Object ob; // ob is now of type Object
    // Pass the constructor a reference to
    // an object of type Object
    NonGen(Object o) {
        ob = o;
    }
    // Return type Object.
    Object getob() {
        return ob;
    }
    // Show type of ob.
    void showType() {
        System.out.println("Type of ob is " +
            ob.getClass().getName());
    }
}
```

De ce nu Object.

```
class NonGenDemo {  
public static void main(String args[]) {  
    NonGen iOb;  
    // Create NonGen Object and store  
    // an Integer in it. Autoboxing still  
    occurs.  
    iOb = new NonGen(88);  
    // Show the type of data used by iOb.  
    iOb.showType();  
    // Get the value of iOb.  
    // This time, a cast is necessary.  
    int v = (Integer) iOb.getob();  
    System.out.println("value: " + v);  
    System.out.println();  
    // Create another NonGen object and  
    // store a String in it.    }  
}
```

```
NonGen strOb = new NonGen("Non-  
Generics Test");  
// Show the type of data used by  
strOb.  
strOb.showType();  
// Get the value of strOb.  
// Again, notice that a cast is  
necessary.  
String str = (String) strOb.getob();  
System.out.println("value: " + str);  
// This compiles, but is conceptually  
wrong!  
iOb = strOb;  
v = (Integer) iOb.getob(); // run-time  
error!
```

Generice cu 2 parametri

```
class TwoGen<T, V> {  
    T ob1;  
    V ob2;  
    // Pass the constructor a reference to  
    // an object of type T and an object of type V.  
    TwoGen(T o1, V o2) {  
        ob1 = o1;  
        ob2 = o2;  
    }  
    // Show types of T and V.  
    void showTypes() {  
        System.out.println("Type of T is " +  
            ob1.getClass().getName());  
        System.out.println("Type of V is " +  
            ob2.getClass().getName());  
    }  
    T getob1() {  
        return ob1;  
    }  
    V getob2() {  
        return ob2;  
    }  
}
```

Demo generice cu 2 parametri

```
class SimpGen {  
public static void main(String args[]) {  
    TwoGen<Integer, String> tgObj =  
        new TwoGen<Integer, String>(88, "Generics");  
    // Show the types.  
    tgObj.showTypes();  
    // Obtain and show values.  
    int v = tgObj.getob1();  
    System.out.println("value: " + v);  
    String str = tgObj.getob2();  
    System.out.println("value: " + str);  
}  
}
```

Limite

```
class Stats<T> {  
    T[] nums; // nums is an array of type T  
    // Pass the constructor a reference to  
    // an array of type T.  
    Stats(T[] o) {  
        nums = o;  
    }  
    // Return type double in all cases.  
    double average() {  
        double sum = 0.0;  
        for(int i=0; i < nums.length; i++)  
            sum += nums[i].doubleValue(); // Error!!!  
        return sum / nums.length;  
    }  
}
```

Solutie

```
class Stats<T extends Number> {  
    T[] nums; // nums is an array of type T  
    // Pass the constructor a reference to  
    // an array of type T.  
    Stats(T[] o) {  
        nums = o;  
    }  
    // Return type double in all cases.  
    double average() {  
        double sum = 0.0;  
        for(int i=0; i < nums.length; i++)  
            sum += nums[i].doubleValue(); // correct  
        return sum / nums.length;  
    }  
}
```

Demo

```
// Demonstrate Stats.

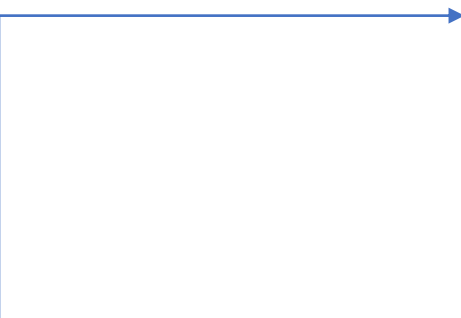
class BoundsDemo {

public static void main(String args[]) {
    Integer inums[] = { 1, 2, 3, 4, 5 };
    Stats<Integer> iob = new Stats<Integer>(inums);
    double v = iob.average();
    System.out.println("iob average is " + v);
    Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    Stats<Double> dob = new Stats<Double>(dnums);
    double w = dob.average();
    System.out.println("dob average is " + w);
    // This won't compile because String is not a
    // subclass of Number.
    // String strs[] = { "1", "2", "3", "4", "5" };
    // Stats<String> strob = new Stats<String>(strs);
    // double x = strob.average();
    // System.out.println("strob average is " + v);
}

}
```

Wildcard - ?

```
class Stats<T extends Number> {  
    T[] nums; // nums is an array of type T  
    // Pass the constructor a reference to  
    // an array of type T.  
    Stats(T[] o) {  
        nums = o;  
    }  
    // Return type double in all cases.  
    double average() {  
        double sum = 0.0;  
        for(int i=0; i < nums.length; i++)  
            sum += nums[i].doubleValue();  
        return sum / nums.length;  
    }  
    // Determine if two averages are the same.  
    // Notice the use of the wildcard.  
    boolean sameAvg(Stats<?> ob) {  
        if(average() == ob.average())  
            return true;  
        return false;  
    }  
}
```



Orice tip de data independent de T

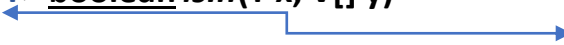
Demo

```
class WildcardDemo {  
    public static void main(String args[]) {  
        Integer inums[] = { 1, 2, 3, 4, 5 };  
        Stats<Integer> iob = new Stats<Integer>(inums);  
        double v = iob.average();  
        System.out.println("iob average is " + v);  
        Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };  
        Stats<Double> dob = new Stats<Double>(dnums);  
        double w = dob.average();  
        System.out.println("dob average is " + w);  
  
        Float fnums[] = { 1.0F, 2.0F, 3.0F, 4.0F, 5.0F };  
        Stats<Float> fob = new Stats<Float>(fnums);  
        double x = fob.average();  
  
        System.out.println("fob average is " + x);  
        // See which arrays have same average.
```

```
        System.out.print("Averages of iob and dob ");  
        if(iob.sameAvg(dob))  
            System.out.println("are the same.");  
        else  
            System.out.println("differ.");  
        System.out.print("Averages of iob and fob ");  
        if(iob.sameAvg(fob))  
            System.out.println("are the same.");  
        else  
            System.out.println("differ.");  
    }  
}
```

Metode generice

```
class GenMethDemo {  
    // Determine if an object is in an array.  
    static <T extends Comparable<T>, V extends T> boolean isIn(T x, V[] y)  
    {  
        for(int i=0; i < y.length; i++)  
            if(x.equals(y[i])) return true;  
        return false;  
    }  
  
    public static void main(String args[]) {  
        // Use isIn() on Integers.  
        Integer nums[] = { 1, 2, 3, 4, 5 };  
        if(isIn(2, nums))  
            System.out.println("2 is in nums");  
        if(!isIn(7, nums))  
            System.out.println("7 is not in nums");  
        System.out.println();  
        // Use isIn() on Strings.  
        String strs[] = { "one", "two", "three",  
            "four", "five" };  
        if(isIn("two", strs))  
            System.out.println("two is in strs");  
        if(!isIn("seven", strs))  
            System.out.println("seven is not in strs");  
        // Oops! Won't compile! Types must be compatible.  
        // if(isIn("two", nums))  
        // System.out.println("two is in strs");  
    }  
}
```



Constructori generici

```
class GenCons {  
  
    private double val;  
  
    <T extends Number> GenCons(T arg) {  
        val = arg.doubleValue();  
    }  
  
    void showval() {  
        System.out.println("val: " + val);  
    }  
}
```

```
class GenConsDemo {  
  
    public static void main(String args[]) {  
        GenCons test = new GenCons(100);  
        GenCons test2 = new GenCons(123.5F);  
        test.showval();  
        test2.showval();  
    }  
}
```

Interfete generice

```
interface MinMax<T extends Comparable<T>> {
    T min();
    T max();
}

// Now, implement MinMax
class MyClass<T extends Comparable<T>> implements MinMax<T> {
    T[] vals;

    MyClass(T[] o) {
        vals = o;
    }

    // Return the minimum value in vals.
    public T min() {
        T v = vals[0];
        for (int i = 1; i < vals.length; i++) {
            if (vals[i].compareTo(v) < 0) {
                v = vals[i];
            }
        }
        return v;
    }
}
```



```
public T max() {
    T v = vals[0];
    for (int i = 1; i < vals.length; i++) {
        if (vals[i].compareTo(v) > 0) {
            v = vals[i];
        }
    }
    return v;
}

class GenIFDemo {

    public static void main(String args[]) {
        Integer inums[] = {3, 6, 2, 8, 6};
        Character chs[] = {'b', 'r', 'p', 'w'};
        MyClass<Integer> iob = new MyClass<Integer>(inums);
        MyClass<Character> cob = new MyClass<Character>(chs);
        System.out.println("Max value in inums: " + iob.max());
        System.out.println("Min value in inums: " + iob.min());
        System.out.println("Max value in chs: " + cob.max());
        System.out.println("Min value in chs: " + cob.min());
    }
}
```

Folosind perechi

```
class Pair<K, Integer> {  
    @Override  
    public String toString() {  
        return "value = " + val + " index = " + index;  
    }  
    public K val;  
    public Integer index;  
}  
  
interface MinMax<T extends Comparable<T>, Integer> {  
    Pair<T, Integer> min();  
    Pair<T, Integer> max();  
}  
  
// Now, implement MinMax  
class MyClass<T extends Comparable<T>> implements MinMax<T, Integer> {  
    T[] vals;  
    MyClass(T[] o) {  
        vals = o;  
    }  
    // Return the minimum value in vals.  
    public Pair<T, Integer> min() {  
        Pair<T, Integer> v = new Pair<T, Integer>();  
        v.val = vals[0];  
        v.index = 0;  
    }  
}
```

```
        for (int i = 1; i < vals.length; i++) {  
            if (vals[i].compareTo(v.val) < 0) {  
                v.index = i;  
                v.val = vals[i];  
            }  
        }  
        return v;  
    }  
    // Return the maximum value in vals.  
    public Pair<T, Integer> max() {  
        Pair<T, Integer> v = new Pair<T, Integer>();  
        v.val = vals[0];  
        v.index = 0;  
        for (int i = 1; i < vals.length; i++) {  
            if (vals[i].compareTo(v.val) > 0) {  
                v.index = i;  
                v.val = vals[i];  
            }  
        }  
        return v;  
    }  
}
```

Mostenirea unei clase generice

```
class Gen<T> {  
    T ob;  
    Gen(T o) {  
        ob = o;  
    }  
    // Return ob.  
    T getob() {  
        return ob;  
    }  
}  
// A subclass of Gen.  
class Gen2<T> extends Gen<T> {  
    Gen2(T o) {  
        super(o);  
    }  
}
```

Exemplu mostenire

```
class Gen<T> {
    T ob;

    Gen(T o) {
        ob = o;
    }

    // Return ob.
    T getob() {
        return ob;
    }
}

class Gen2<T, V> extends Gen<T> {
    V ob2;

    Gen2(T o, V o2) {
        super(o);
        ob2 = o2;
    }

    V getob2() {
        return ob2;
    }
}
```

```
class HierDemo {
    public static void main(String args[]) {
        // Create a Gen2 object for String and Integer.
        Gen2<String, Integer> x
            = new Gen2<String, Integer>("Value is: ", 99);
        System.out.println(x.getob());
        System.out.println(x.getob2());
    }
}
```

Mostenire a unei clase non-generice

```
class NonGen {  
    int num;  
    NonGen(int i) {  
        num = i;  
    }  
    int getnum() {  
        return num;  
    }  
}
```

```
class Gen<T> extends NonGen {  
    T ob;  
    Gen(T o, int i) {  
        super(i);  
        ob = o;  
    }  
    T getob() {  
        return ob;  
    }  
}
```

```
class HierDemo2 {  
    public static void main(String args[]) {  
        // Create a Gen object for String.  
        Gen<String> w = new Gen<String>("Hello", 47);  
        System.out.print(w.getob() + " ");  
        System.out.println(w.getnum());  
    }  
}
```

Ambiguitati

```
// Ambiguity caused by erasure on
// overloaded methods.

class MyGenClass<T, V> {

    T ob1;

    V ob2;

    // These two overloaded methods are ambiguous
    // and will not compile.

    void set(T o) {
        ob1 = o;
    }

    void set(V o) {
        ob2 = o;
    }
}
```

Restrictii

// Can't create an instance of T.

```
class Gen<T> {  
    T ob;  
    Gen() {  
        ob = new T(); // Illegal!!!  
    }  
}
```

//can't extend Throwable

```
class Wrong<T> {
```

// Wrong, no static variables of type T.

```
    static T ob;
```

// Wrong, no static method can use T.

```
    static T getob() {
```

```
        return ob;
```

```
    }
```

```
}
```

Restrictii - array

```
class Gen<T extends Number> {

    T ob;
    T vals[]; // OK

    Gen(T o, T[] nums) {
        ob = o;
        // This statement is illegal.
        // vals = new T[10]; // can't create an
        // array of T
        // But, this statement is OK.
        vals = nums; // OK to assign
        // reference to existent array
    }
}
```

```
class GenArrays {

    public static void main(String args[])
    {
        Integer n[] = {1, 2, 3, 4, 5};
        Gen<Integer> iOb = new
        Gen<Integer>(50, n);
        // Can't create an array of type-specific
        // generic references.
        // Gen<Integer> gens[] = new
        // Gen<Integer>[10]; // Wrong!
        // This is OK.
        Gen<?> gens[] = new Gen<?>[10]; //
        OK
    }
}
```

Exercitii

- Implementati o clasa ce sa permita cautarea pozitiei unei valori date, intr-un sir generic. Indiciu:
 - *<T extends Comparable> int Search(T[] a, T key)*
- Se da mai jos clasa Pair
 - Modificati clasa astfel ca ambii membri sa aiba acelasi tip
 - Adaugati o metoda *swap* care sa interschimbe cei doi membri
 - Adaugati o metoda statica PairUtil.minmax care sa calculeze atat maximul cat si minimul dintr-un sir de elemente de tip T. Aceasta metoda va returna un obiect Pair ce contine valorile max si min.
- Implementati o versiune de generare a unui sir generic ce contine permutarile a n elemente, unde n se citeste de la tastatura.
- Pentru exercitiile de mai sus, tratati exceptiile ce pot aparea.

Pair

```
public class Pair<T, S>
{
    public Pair(T firstElement, S secondElement)
    {
        first = firstElement;
        second = secondElement;
    }

    public T getFirst() { return first; }
    public S getSecond() { return second; }
    private T first;
    private S second;
}
```