

Iterator & for cyklus

- for (Object o : foo)
 - funguje pokud je foo pole nebo **je foo iterovatelné**
 - jako to zařídit?
 - implementovat interface **java.lang.Iterable**
- **Iterable** má jednu metodu
java.util.Iterator iterator()
- metody interfacu **Iterator**
 - boolean hasNext()
 - Object next()
 - void remove()
- iterator se typicky implementuje pomocí anonymní vnitřní třídy
- ve skutečnosti je Iterator generický typ, tj. **Iterator<T>**
 - zatím to budeme ignorovat...

iterator příklad

```
public class MyArrayWithIterator implements Iterable {  
    private Object[] arr = new Object [5];  
    private int s = 0;
```

```
    public int size() {  
        return s;  
    }
```

```
    public void add(Object o) {  
        ...  
        arr[s++] = o;  
        ...  
    }
```

```
    public Iterator iterator() {  
        return new Iterator() {  
            private int index = 0;  
            public boolean hasNext() {  
                return index < s;  
            }  
            public Object next() {  
                return arr[index++];  
            }  
            public void remove() {  
                throw new  
                UnsupportedOperationException();  
            }  
        };  
    }
```

- od Java 8 je remove() **default**
- implementace vyhazuje tuto výjimku

Úkol 1

- vytvořte interface MyCollection s metodami
 - void add(Object o)
 - Object get(int i)
 - void remove(Object o)
 - void remove(int i)
 - int size()
- vytvořte implementaci interfacu MyCollection
 - použijte pole, které se podle potřeby realokuje
 - chybové stavy ošetřete výjimkami
 - při přístupu mimo rozsah pole
- přidejte k implementaci iterator (viz předchozí 2 slidy)

Úkol 2

- napište třídu která reprezentuje vyvážený binární vyhledávací strom (třeba AVL, RB, nebo jakýkoliv jiný)
 - pro typ `int`
- přidejte ke třídě iterator, který projde strom od nejmenšího prvku k největšímu
- napište program který strom používá a data do něj načte z parametrů příkazové řádky
 - použijte `Integer.parseInt(String s)` pro převod z `Stringu` do `intu`
 - nezapomeňte ošetřit případné výjimky, které metoda vyhodí, pokud řetězec nelze převést na `int`
- zamyslete jak upravit strom, aby nebylo nutné ho napsat pro konkrétní typ, ale obecně pro `Object`
 - tj. jak zařídit, aby vkládané prvky šly porovnávat
 - naimplementujte

Testy...

Test 1

- Co program vypíše – true nebo false

```
public class Test01 {
    public static void main(String[] argv) {
        System.out.println(test());
    }

    public static boolean test() {
        try {
            return true;
        } finally {
            return false;
        }
    }
}
```

Test 2

- Co program vypíše?

```
public class Test02 {  
  
    public static void main(String[] argv) {  
        try {  
            System.out.println("Hello world!");  
            System.exit(0);  
        } finally {  
            System.out.println("Goodbye");  
        }  
    }  
}
```

Test 3

- Co se vypíše

```
public class ParamsTest {
    public ParamsTest(Object o) {
        System.out.println("ParamsTest(Object o)");
    }
    public ParamsTest(long[] a) {
        System.out.println("ParamsTest(long[] a)");
    }
    public static void main(String[] argv) {
        new ParamsTest(null);
    }
}
```

- A nelze přeložit
- B ParamsTest(Object o)
- C ParamsTest(long[] a)

Test 3

- C je správně
- Proč?
 - Postup hledání metody/konstruktoru
 - podle skutečných parametrů se vybere, co je použitelné
 - z vybraných metod/konstruktorů se podle typů **formálních** parametrů vybere nejspecifičtější metoda/konstruktor
 - **ParamsTest(long[] a)** je specifičtější než **ParamTest(Object o)**
 - vše, co lze předat do **long[] a**, lze předat i do **Object**
 - obráceně to ale neplatí

Test 4



ZKOUŠKOVÝ PŘÍKLAD

- Co se vypíše

```
class A {
    public static void foo() {
        System.out.println("foo");
    }
}
class B extends A {
    public static void foo() {
        System.out.println("bar");
    }
}
```

```
public class OverloadTest {
    public static void
        main(String[] argv) {
        A a = new A();
        A b = new B();
        a.foo();
        b.foo();
    }
}
```

- A foo bar
- B foo foo
- C bar bar
- D něco jiného

Test 4

- B je správně
- statické metody nejsou virtuální



Verze prezentace PJ03.cz.2019.01

Tato prezentace podléhá licenci [Creative Commons Uveďte autora-Neužívejte komerčně 4.0 Mezinárodní License](https://creativecommons.org/licenses/by-nc/4.0/).