6. Gyakorlat

legendi@inf.elte.hu

2010. március 16.

Kiegészítés

- Oprendszer függő tulajdonságok:

```
// PATH elvalaszto, pl. ":" - Unix, ";" - Windows
final String PATH_SEPARATOR = File.pathSeparator;
// Ugyanaz, csak karakterkent
final char PATH_CHAR = File.pathSeparatorChar;

// Nev szeparator, pl. "/" - Unix, "\" - Windows
final String SEPARATOR = File.separator;
// Ugyanaz, csak karakterkent
final char SEPARATOR_CHAR = File.separatorChar;

// Sorvege karakter, pl. "\n" - Unix, "\r\n" - Windows
// Reszletesen lasd FAQ!
final String EOL = System.getProperty("line.separator");
```

- ++i, i++

```
int i = 0;
System.out.println(i++); // kiir, megnovel: "0"
System.out.println(++i); // megnovel, kiir: "2"
```

- **Közvetlen elérésű fájlok** A `java.io.RandomAccessFile`, kb. mint egy bájtvektor, olvasható és írható. Fájlmutató az aktuális pozícióra, ez lekérdezhető(`getFilePointer()`), állítható (`seek(int)`). Implementálja mind a DataInput, DataOutput interfészeket (mindkettőt egyszerre), a műveleteivel tetszőleges típus írható, olvasható (úgy használható, mint a DataInputStream, DataOutputStream: `write*`, `read*` függvények), byte-ok átugorhatók (`skip(int)`). Példaalkalmazás:

```
package gyak6;
```

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomFileTest {
    public static void main(final String[] args)
            throws IOException {
        final RandomAccessFile raf =
            new RandomAccessFile("dummy.dat", "rw");

        raf.writeInt(0xCAFEBABE);
        raf.seek(16);
        raf.writeInt(0xDEADBEEF);
        raf.seek(32);
        raf.writeInt(0xBADF00D0);
        raf.seek(48);
        raf.writeInt(0xDEADC0DE);
        raf.close();
    }
}
```

Megjegyzés: bájtokat írunk ki, nem elfelejteni! Hexa módban kell megnyitni a fájlt, hogy lássuk ténylegesen mi is íródott ki, pl. TC + F3 + 3, vagy vi + ":%!xxd".

Gotchaz:

- write(int) - hiába van int paramétere, csak byte-ot ír ki, a legalját
- seek(), write(...) - nem tolódik tovább a stream, kézzel kell mindent odébbmásolni
- raf.seek(file.length()); - file végére ugrás, ahhoz hozzáfűzés
- write() - elfogad byte[] paramétert, de writeBytes() - csak Stringet
- EOFException - elindexelésnél (IOException leszármazottja)

Részletesen: `http://java.sun.com/javase/6/docs/api/java/io/RandomAccessFile.html`

- **Listenerek egyéb megvalósításai** Saját listener megvalósítása:

```
public class GUI1 {
    private final JButton ok = new JButton("Ok");

    private class OkButtonActionListener implements ActionListener {
        @Override
```

```
        public void actionPerformed(ActionEvent e) {
            System.out.println("Ok pressed");
        }
    }

    public GUI1() {
        OkButtonActionListener listener = new OkButtonActionListener();
        ok.addActionListener(listener);
    }
}
```

vagy maga az osztály implementálja az interfészt, mindenre rá lehet aggatni, majd setActionCommand() ill. getActionCommand() függvény használható a megkülönböztetésre:

```
public class GUI2 implements ActionListener {
    private final JButton ok = new JButton("Ok");
    private final JButton cancel = new JButton("Ok");

    private final String OK_COMMAND     = "ok";
    private final String CANCEL_COMMAND = "cancel";

    public GUI2() {
        ok.addActionListener(this);
        ok.setActionCommand(OK_COMMAND);

        cancel.addActionListener(this);
        cancel.setActionCommand(CANCEL_COMMAND);
    }

    public void actionPerformed(ActionEvent e) {
        if (OK_COMMAND.equals(e.getActionCommand())) {
            System.out.println("Ok pressed");
        } else if (CANCEL_COMMAND.equals(e.getActionCommand())) {
            System.out.println("Cancel oressed");
        }
    }
}
```

- **Komplexebb grafikus felület** Containerekkel (Panel, JPanel).

Feladat

Készítsünk egy egyszerű konzolos alkalmazást, amely képes fájlok bináris karbantartására! A program első paraméterként kapja meg a szerkesztendő fájl nevét. A további paraméterek a következők lehetnek:

1. 'mb <pozíció> <byte>': az adott pozíción lévő byte érték módosítása
2. 'mi <pozíció> <int>':az adott pozíción lévő int érték módosítása
3. 'i <pozíció> <szöveg>': a specifikált szöveg beillesztése az adott pozícióra. A program illessze be a szöveget, azaz tolja el a bájtokat megfelelőképp.

Annotációk

Dekorációk a forráskódban, első sorban külső toolok számára hasznos eszközök, de pl. a fordító, ill. maga a program is hasznukat veheti (voltak ad hoc jellegű megfelelelőik eddig is, pl. @deprecated javadoc tag). Általános célú eszköz, Java 5.0 óta, metainformációt közölhetnek.

A program szemantikájára direkt módon nincsenek hatással, viszont különböző eszközök, libek ezt az információt már felhasználhatják a program futásának módosítására. Kiegészítik a javadoc tageket.

Felhasználási lehetőségek:

- Információ a fordítónak (pl. warningok kikapcsolása, elavult kódrészletek jelzése)
- Fordítási, deployment információk (pl. kódgenerálás)
- Futásidejű feldolgozás (pl. egyes annotációk futási időben is elérhetők)

Fontosabb beépített annotációk

Megjegyzés: @ = "AT", mint "Annotation Type"

@Override Felüldefiniált metódusok jelzésére, fordítási idejű ellenőrzés

```
@Override
public String toString() { ... }
```

@Deprecated Elavult, ám reverse compatibility miatt fontos függvények jelölésére. Fordítási idejű ellenőrzés, warningot generál.

```
@Deprecated
public void someChaoticMethod { ... }
```

@SuppressWarnings Adott kódrészletben a fordítási idejű figyelmeztetések kikapcsolása (kódrészlet = TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE). Opciók lehetnek: "deprecation", "unchecked", "unused".

```
@SuppressWarnings("deprecation")
public int someChaoticFunction() { ... }

@SuppressWarnings({ "deprecation", "unchecked" }) // ld. kesobb
public int someVeryChaoticFunction() { ... }
```

Definiálás

Kiterjeszthető: saját változatokat is lehet definiálni, `@interface` kulcsszó. Paraméter nélkül *marker*:

```
@interface MayBeNull {}

class PersonalData {
    @MayBeNull private String maidenName;
}
```

Ha egyetlen értéke van, azt érdemes *value()*-nak hívni, mert rövidebb használni:

```
@interface MayBeNull {
    String value();
}

class PersonalData {
    @MayBeNull("if (gender == male)")
    String private maidenName;
}
```

Ha a value() String[] típusú, akkor használható simán "" vagy { "", "" } forma is:

```
@interface MayBeNull {
    String[] value();
}

class PersonalData {
    @MayBeNull("if (gender == male)")
    private String maidenName;
```

```
    @MayBeNull({"agreed to term of usage", "specified value"})
    private int salary;
}
```

Különben ki kell írni az annotáció használatánál a *tag = érték* párokat:

```
@interface MayBeNull {
    String description();
}

class PersonalData {
    @MayBeNull(description = "if (gender == male)")
    private String maidenName;
}
```

Több tag is megadható, vesszővel elválasztva. Alapértelmezett érték is definiálható:

```
@interface MayBeNull {
    String description();
    boolean managed() default false;
}

class PersonalData {
    @MayBeNull(description = "if (gender == male)")
    private String maidenName;
}
```

Megszorítások

1. Nem lehet generikus – ld. később :-)
2. A függvények
   (a) sem lehetnek generikusak
   (b) nem lehetnek paraméterei
   (c) nem tartalmazhatnak throws deklarációt
   (d) visszatérési értékük csak a következő lehet: primitív típus, String, enum, Class, annotáció (ciklikus hivatkozás szintén tilos), ezekből képzett 1 dimes tömb.
3. nem lehet szülőinterfésze, de implicit módon kiterjeszti a `java.lang.Annotation` osztályt. Metódusai nem ütközhetnek sem az ebben, sem az Object-ben definiált metódusokkal.
4. de: mint az interfészek, tartalmazhatnak osztály, interfész, enum, etc. definíciókat.

Meta-annotációk

A API-ban a `java.lang.annotation.*` csomag

@Retention() Annotáció hozzáférhetőségének szabályozása, java.lang

1. `RetentionPolicy.SOURCE` - csak forráskódban látható, fordításnál kiesik (mint a comment)
2. `RetentionPolicy.RUNTIME` - futtatási időben is hozzáférhető
3. `RetentionPolicy.CLASS` - a class file-ba belekerül, de a JVM nem fér hozzá

@Target() Annotáció használhatóságának szabályozása, `java.lang.annotation.ElementType` használatával: ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE, TYPE_PARAMETER, TYPE_USE

@Inherited Kizárólag osztálydefinícióra, származtatásnál az adott annotáció is öröklődik

@Documented Bekerül a javadoc-kal generált API leírásba is

Példa

```
@Retention(RetentionPolicy.SOURCE)
@Target( { ElementType.FIELD, ElementType.PARAMETER,
           ElementType.LOCAL_VARIABLE } )
@interface MayBeNull {
    String value();
}
```

Felhasználás

```
@SuppressWarnings("deprecation")
public void deprecatedFunction() {
    JFrame frame = new JFrame();
    frame.show(); // deprecated
}

@SuppressWarnings("unchecked")
public void supressedFunction() {
    Vector v = new Vector(); // warning
}
```

Futási időben való elemzés: később, a reflection tárgyalásánál.

Feladat

Készíts egy saját @WrittenBy annotációt, amely tartalmazza az adott osztály, függvény szerzőjének nevét (author, amely alapértelmezetten a te nevedet tartalmazza), az utolsó módosítás dátumát (szintén egy stringben), valamint a verziószámot, amely egy double (és alapértelmezett értéke 1.0). Gondoskodj róla, hogy az adott annotációt csak osztály és függvénydefiníció esetén lehessen alkalmazni, valamint hogy kerüljön bele a generált dokumentációba!

Generic

Egyszerűbb példák (java.util csomagból):

```
public interface List<E> {
    void add(E x);
    Iterator<E> iterator();
}

public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

E - formális típusparaméter, amely aktuális értéket a kiértékelésnél vesz fel (pl. Integer, etc.).

Altípusosság

Nem konvertálhatók, ennek oka:

```
List<String> l1 = new ArrayList<String>();
List<Object> l2 = l1; // error

// Mert akkor lehetne ilyet csinalni:
l2.add(new Object());
l1.get(0); // reccs, Object -> String castolas
```

Magyarul ha $S \sqsubseteq T \nRightarrow G\langle S\rangle \sqsubseteq G\langle T\rangle$ - ez pedig ellent mond az ember megérzésének. Castolni lehet (warning), instanceof tilos (fordítási hiba)!

Wildcardok

Probléma: általános megoldást szerenténk, amely minden collectiont elfogad, függetlenül az azokban tárolt elemektől (pl. ki szeretnénk őket írni),

vagy nem tudjuk azok konkrét típusát (pl. legacy code). Collection<Object> *nem* őse (ld. előző bekezdés). Ha nem használunk generic-eket, megoldható, viszont warningot generál:

```
void print(Collection c) {
    for (Object o : c) System.out.println(o);
}
```

A megoldás a wildcard használata: Collection<?> minden kollekcióra ráillik. Ilyenkor Objectként hivatkozhatunk az elemekre:

```
void print(Collection<?> c) {
    for (Object o : c) System.out.println(o);
}
```

Vigyázat! A ? $\neq$ *Object*! Csak egy ismeretlen típust jelent. Így a következő kódrészlet is fordítási hibához vezet:

```
List<?> c = ...;
l.add(new Object()); // forditasi hiba
```

Nem tudjuk, hogy mi van benne, lekérdezni viszont lehet (mert tudjuk, hogy minden objektum az Object leszármazottja).

Bounded wildcard

Amikor tudjuk, hogy adott helyen csak adott osztály leszármazottai szerepelhetnek, első (rossz) megközelítés:

```
abstract class Super {}
class Sub1 extends Super {}
class Sub2 extends Super {}
...
void func(List<Super> l) {...} // Rossz!
```

Probléma: func() csak List<Super>-rel hívható meg, List<Sub1>, List<Sub2> nem lehet paramétere (nem altípus). Megoldás: bounded wildcard:

```
void func(List<? extends Super> l) {...}
```

Belepakolni ugyanúgy nem tudunk, mint a ? esetén, azaz erre fordítási hibát kapunk:

```
void func(List<? extends Super> l) {
    l.add(new Sub1()); // reccs
}
```

Generikus osztályok, függvények

Osztálydefinícióban bevezethető típusparaméter az osztályhoz, ez minden membernél használható. Példa:

```
package gyak6;

public class Pair<T, S> {
    private final T first;
    private final S second;

    public Pair(final T first, final S second) {
        super();
        this.first = first;
        this.second = second;
    }

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

    @Override
    public String toString() {
        return "(" + first + ", " + second + ")";
    }
}
```

Generikus függvények esetén szintén a definícióban használható. Példa:

```
package gyak6;

public class ArrayUtils {
    public static final <T, S extends T>
    boolean isIn(final T[] arr, final S element) {
        for (final T t : arr) {
            if (t.equals(element)) return true;
        }

        return false;
    }

    public static void main(final String[] args) {
```

```
        final String[] sarr = {"a", "b", "c"};
        System.out.println( isIn(sarr, "c") );
    }
}
```

Részletesen: `http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf`

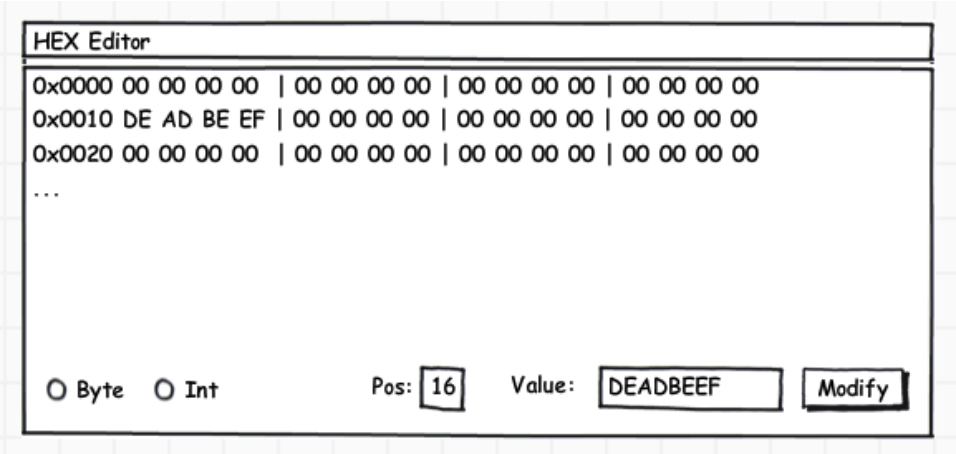
Feladatok

Generikus bináris keresőfa

Készítsünk egy egyszerű, általános bináris keresőfa implementációt! A fához lehessen elemet hozzáadni (add()), kiírni, valamint a minimum, maximum elemet megkeresni (min(), max()). A típusparaméterének összehasonlíthatónak kell lennie (< T extends Comparable<T> >).

Hex editor

Készíts egy egyszerű hexadecimális szövegszerkesztőt! Egy sorban 16 karakter legyen, amely az egyes bájtok értékeit reprezentálják. A program kínáljon lehetőséget byte, int módosításra, adott pozíción, megadott értékkel. A szövegmezőn keresztüli szerkesztés legyen letiltva (`setEnabled(false)`). Ha megnyomják a modify gombot, akkor az adott értéket írjuk ki a file-ba (RandomAccessFile segítségével) a megadott pozícióra, és frissítsük a szövegmező tartalmát!



1. ábra. Képernyőterv