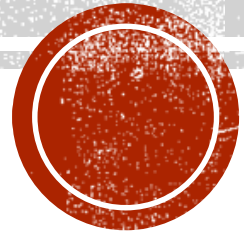
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Генерички тип



ГЕНЕРИЧКИ ТИП

- Генерички тип омогућава употребу промењљиве на месту где се користи назив типа података (класе).
- Као што логичка променљива узима вредност из скупа {**true**, **false**} тако генерички тип узима вредност из скупа класа или интерфејса.
- Један од главних мотива за увођење генеричких типова је развој генеричких алгоритама који се могу примењивати на различите типове података.
- Као добар пример за генеричку функцију може да послужи C-функција **qsort**() помоћу које се реализује уопштен алгоритам за сортирање.
  - Ова функција као аргументе прихвата низ који се сортира, број његових елемената, величину елемента и показивач на функцију поређења.
  - Последња два аргумента омогућавају прилагођавање уопштеног алгоритма за сортирање конкретном типу података.

СИРОВИ ТИП

- У досадашњем тексту користили су се искључиво сирови типови.
  - То су на пример: Integer, Float, LocalDateTime, String итд.
- Њихова употреба је у већини ситуација довољна,
  посебно за писање програма који имају конкретну намену.
- Са друге стране, ако је потребно писати програмске библиотеке општије намене сирови типови могу бити неадекватни.
- Програми опште намене писани су у Јави и пре појаве генеричких типова, коришћењем класе Object, али су били подложни грешкама.

ПРИМЕР 1

- Написати Јава програм за рад са класом која представља кутију.
- У кутију је могуће убацити произвољни објекат.
- Дефинисати методе за убацивање и извлачење објекта из кутије и тестирати њихов рад.

ПРИМЕР 1 (2)

```
public class KutijaSirova {
    // можемо убацити било који објектни тип овде
    private Object vrednost;

    // било који објектни тип се прихвата и имплицитно конвертује у Object
    void postaviVrednost(Object vrednost) {
        this.vrednost=vrednost;
    }

    // изгубили смо информацију о правој природи објектног типа тако да враћамо тип Object
    Object uzmiVrednost() {
        return this.vrednost;
    }

    public static void main(String[] args) {
        KutijaSirova kutija1 = new KutijaSirova();
        kutija1.postaviVrednost("Текст");
        // мора експлицитна конверзија
        String tekst1 = (String) kutija1.uzmiVrednost();
        System.out.println(tekst1);
        // тако да је одговорност на програмеру да зна шта је у кутији
        Integer broj1 = (Integer) kutija1.uzmiVrednost(); // изузетак
    }
}
```

ПОЈАМ, ДЕФИНИСАЊЕ И ПРЕДНОСТИ ГЕНЕРИЧКОГ ТИПА

- Генерички типови се понашају слично као параметри при дефинисању метода.
- Програмирање коришћењем генеричких типова има следеће предности:
  - строжија контрола типова приликом превођења Јава програма;
  - елиминација експлицитне конверзије типова (кастовања);
  - могућност имплементације генеричких алгоритама.
- Генеричка класа се дефинише на следећи начин:

```
imeKlase<T1,T2,...,Tn>{/* ... */}
```

- По конвенцији, параметри су означени једним великим словом. Најчешће ознаке параметара су:
  - **E** – елеменат (енг. **element**)
  - **K** – кључ (енг. **key**)
  - **N** – број (енг. **number**)
  - ...

ПРИМЕР 2

- Написати Јава програм за рад са класом која представља кутију попут оне у примеру 1.
- За разлику од примера 1, класу је потребно реализовати као генеричку.

ПРИМЕР 2 (2)

```
public class KutijaGenericka<T> {
    // можемо убацити било који објектни тип овде
    private T vrednost;

    // прихвата се онај објектни тип за кога је направљена кутија
    public void postaviVrednost(T vrednost) {
        this.vrednost=vrednost;
    }

    // нисмо изгубили информацију о правој природи објектног типа
    public T uzmiVrednost() {
        return this.vrednost;
    }

    public static void main(String[] args) {
        KutijaGenericka<String> kutija1 = new KutijaGenericka<String>();
        kutija1.postaviVrednost("Текст");
        // не треба експлицитна конверзија
        // тако да је програмер растерећен одговорности да то мора да зна
        String tekst1 = kutija1.uzmiVrednost();
        System.out.println(tekst1);
        // компајлер не допушта да у кутију убацимо нешто што није String
        //kutija1.postaviVrednost(45);
    }
}
```

ПРИМЕР 3

- Написати Јава програм за рад са генеричком класом која представља генерички уређени пар вредности произвољних типова.
- Поред конструктора, који прихвата обе вредности, потребно је реализовати и методе за дохватање првог и другог члана уређеног пара.
- Такође је потребно редефинисати метод **toString()**.

ПРИМЕР 3 (2)

```
public class UredjeniPar<T, S>{
    private T vrednost1;
    private S vrednost2;

    public UredjeniPar(T vrednost1, S vrednost2) {
        this.vrednost1 = vrednost1;
        this.vrednost2 = vrednost2;
    }

    public T getVrednost1() {
        return vrednost1;
    }

    public S getVrednost2() {
        return vrednost2;
    }

    @Override
    public String toString() {
        return "("+vrednost1+", "+vrednost2+")";
    }

    public static void main(String[] args) {
        UredjeniPar<Integer, Integer> par1 =
                new UredjeniPar<Integer, Integer>(10, 20);
        UredjeniPar<Integer, String> par2 = new UredjeniPar<>(30, "Пример текст");
        //UredjeniPar<Integer, Integer> par3 = new UredjeniPar<>(30, 14.0);
        System.out.println(par1);
        System.out.println(par2);
    }
}
```

ГЕНЕРИЧКИ ИНТЕРФЕЈСИ И ЊИХОВА ИМПЛЕМЕНТАЦИЈА

- Генерички интерфејси омогућавају декларисање генеричких метода.

```
public interface Interfejs <T>{
    void f(T arg);
    T g();
}
```

- Приликом њихове имплементације од стране класа генеричност се може задржати или изгубити (прецизирањем класе при дефинисању).

```
class ObicnaKlasa implements Interfejs<String>{ ... }
class GenerickaKlasa<T> implements Interfejs<T>{ ... }
```

ПРИМЕР 4

- Написати Јава програм у којем се дефинише генерички интерфејс за стек структуру података.
- Након тога дефинисати класу која имплементира генерички стек интерфејс помоћу низа и тестира његове могућности.

ПРИМЕР 4 (2)

```
interface StekInterfejs<T> {
    boolean jePrazan(); // празан стек
    int velicina(); // број елемената у стеку
    void dodaj(T element); // додај на врх
    T vrh();    // елемент на врху стека
    void ukloni(); // скини елемент са врха
}
```

```
public class StekPrekoNiza<T> implements StekInterfejs<T> {
    private T[] stekNiz;
    private int stekIndeks; // показује на врх стека
    private int kapacitet;
    ...
```

- Остатак решења погледати у књизи.

САМОСТАЛНИ ГЕНЕРИЧКИ МЕТОД

- Опсег генеричког параметра не мора обухватати читаву класу или интерфејс.
- У Јави је могућа и локалнија (самосталнија) употреба генеричких параметара, тј. употреба само у оквиру појединачних метода.

```
public class PretragaNiza {
    public static <T> int pretrazi(T[] niz, T element) {
        for(int i=0; i<niz.length; i++)
            if(niz[i].equals(element))
                return i;
        return -1;
    }

    public static void main(String[] args) {
        Integer[] nizCelih = new Integer[] {2,43,22,11,243,253,64};
        int element1 = 34;
        System.out.printf("Позиција елемента %d је %d.%n", element1,
                pretrazi(nizCelih, element1));
    }
}
```

ОГРАНИЧЕЊА ЗА ТИПОВЕ

- У неким ситуацијама је потребно да генерички параметар испуњава додатне критеријуме како би могао да се користи за реализацију генеричког алгоритма.
- То значи да параметар не може увек представљати произвољну класу као што је то био случај у досадашњим примерима.
  - На пример, за потребе реализације алгоритма бинарне претраге, сортирања, тражења минимума, максимума итд., типови морају бити упоредиви.

```
<T extends TipKojiOgranicava>
```

- То значи да тип који ће конкретизовати параметар **T** мора бити или једнак типу **TipKojiOgranicava** или испод њега у хијерархији типова.
- Тип може истовремено бити ограничен највише једном класом и произвољним бројем интерфејса.

```
<T extends TipKojiOgranicava1 & TipKojiOgranicava2 & ...>
```

ПРИМЕР 6

- Написати Јава програм који реализује статички локално-генерички метод за тражење минималног елемента у низу.
- Након тога применити метод на низ објеката типа **String** и низ објеката раније уведеног генеричког типа уређени пар.
- Потребно је “дорадити” класу за уређени пар тако да имплементира генерички интерфејс **Comparable**.
- Поређење се врши на основу прве координате, а потом на основу друге ако су прве координате једнаке.

ПРИМЕР 6 (2)

- Погледати решење у књизи.

ГЕНЕРИЦИ И ВИРТУЕЛНА МАШИНА

- Кад год се дефинише генерички тип, у позадини (на нивоу виртуелне машине) аутоматски се обезбеђује одговарајући сирови тип за њега.
- Промењљиве које представљају типове су замењене типовима који их ограничавају одозго или **Object** (ако за те промењљиве није било ограничења).
  - На пример, за раније уведену класу **KutijaGenericka<T>** у позадини се креира сирова класа слична класи **KutijaNegenericka** у којој је параметар типа поља vrednost замењен класом **Object**.
- Додатно, компајлер убацује експлицитну конверзију ка одговарајућем типу **T** на свим местима где је то потребно.
- Када се користи бар једно ограничење за параметар **T**, уместо замене параметра типа T класом **Object**, врши се замена првим наведеним ограничавајућим типом.
- За преостала ограничења, ако их има више од једног, у коду се на одговарајућим местима спроводе експлицитне конверзије како би се добио жељени тип.

ПРИМЕР 7

- Написати Јава програм који класу упоредиви уређени пар из примера 6 реализује употребом искључиво сирових типова.
- Слично као у примеру 6 реализовати статички метод за тражење минималног елемента низа и применити га на низ уређених парова.

ПРИМЕР 7 (2)

- Решење погледати у књизи.

ГЕНЕРИЦИ И НАСЛЕЂИВАЊЕ

- У каквој су релацији две генеричка објекта уколико су њихови параметри у релацији наслеђивања?
  - Нпр., ако су параметри типови **Integer** и **Object**, и познато је да је **Integer** потклаcа класе **Object**, да ли је онда и **KutijaGenericka<Integer>** потклаcа **KutijaGenericka<Object>**?
  - Одговор је одречан. Наиме, објекти ових класа нису ни у каквој релацији те се инстанцна променљива типа **KutijaGenericka<Integer>** не може сакрити (уопштити) имплицитном конверзијом иза објектне променљиве типа **KutijaGenericka<Object>**.
- Са друге стране, уколико не посматрамо наслеђивање између параметара, већ између класа које се параметризују (у овом случају **KutijaGenericka<T>**), онда се може успоставити релација наслеђивања.

ПРИМЕР 8

- Написати Јава програм који реализује генеричку потклasу претходно уведене класе **KutijaGenericka<T>**.
- Потклаcа треба да омогући прослеђивање боје кутије као аргумент конструктора.
- Након тога, тестирати имплицитну конверзију између инстанцних променљивих из смера потклaсе ка надкласи.

ПРИМЕР 8 (2)

```
public class KutijaGenerickaObojena<T> extends KutijaGenericka<T>{
    private Color boja;

    public KutijaGenerickaObojena(Color boja){
        super();
        this.boja=boja;
    }

    public Color getBoja() {
        return boja;
    }

    public static void main(String[] args) {
        KutijaGenerickaObojena<String> kutijaObojena =
                new KutijaGenerickaObojena<String>(Color.red);
        kutijaObojena.postaviVrednost("Текст");
        // имплицитна конверзија у општији тип
        KutijaGenericka<String> kutija = kutijaObojena;
        System.out.println(kutija.uzmiVrednost());
    }
}
```

ПИТАЊА И ЗАДАЦИ

- Које су предности употребе генеричких типова?
- Написати Јава програм у којем се дефинише генерички интерфејс за ред структуру података. Након тога дефинисати класу која имплементира генерички ред интерфејс помоћу низа и тестира његове могућности.
- Шта је локално-генерички метод и када се користи? Илустровати примером.
- Шта се подразумева под ограничењем параметарског типа Т са “горње стране”? Које интерпретације ограничења са “горње стране” постоје?
- Написати Јава програм који реализује статички локално-генерички метод за тражење максималног елемента у низу. Након тога применити метод над низом објеката неке од уграђених класа програмског језика Јава и над низом објеката кориснички дефинисане генеричке класе по избору.

ПИТАЊА И ЗАДАЦИ (2)

- Шта су сирови типови? Примером илустровати употребу сирових типова.
- У каквој су релацији два генеричка објекта уколико су њихови параметри у релацији наслеђивања? Илустровати примером.
- Како се концепт генеричких типова уклапа у концепт наслеђивања? Односно, да ли генеричка класа може да наследи неку другу генеричку класу, да ли може да наследи негенеричку класу? Да ли негенеричка класа може да наследи генеричку класу? Ако је могуће, илустровати примером.
- Да ли је могуће превазићи наслеђени генерички метод из базне класе генеричким методом у изведној класи? Ако је могуће, илустровати примером.