Razvoj softvera

dr.sc. Emir Mešković

IV predavanje

Sadržaj predavanja

- Apstraktne klase i metodi
- Interfejs klase
- Rukovanje iznimkama

Apstraktne klase

- Nekada je potrebno definirati klasu ali onemogućiti instanciranje objekata tipa te klase.
- Za klasu se tada kaže da je **apstraktna** i namjenjena za kreiranje izvedenih klasa.
- C++ omogućava kreiranje apstraktne klase na način da se za neki virtuelni metod u njegovoj deklaraciji nakon definisanja parametara doda =0 .
- Metod se tada naziva čisti virtuelni metod.
- Izvedene klasa da bi postala konkretna (tj. da omogući kreiranje objekata) mora implementirati sve čiste virtuelne metode.
- Iako nije moguće kreirati objekat od apstraktne klase moguće je kreirati pointer na apstraktnu klasu

Primjer


```
#include <iostream>
class GrafObjekat {
    private:
      int x_, y_;
    public:
      GrafObjekat(int x, int y) : x_(x), y_(y) {}
      void pomjeri(int dx, int dy)
      {
          x_ += dx;
          y_ += dy;
      }
      virtual void crtaj()=0;
};
class Krug : public GrafObjekat {
    private:
      int r_;
    public:
      Krug(int r) : GrafObjekat(0,0), r_(r) {}
      void crtaj() { std::cout << "Crtam krug r=" << r_;}
};
int main() {
    //GrafObjekat a(2,3); generira gresku
    GrafObjekat* K = new Krug(10);
    K->pomjeri(5,10);
    K->crtaj();
    delete K;
    return 0;
}
```

Java apstraktne klase

- Da bi Java klasa postala apstraktna potrebno je koristiti ključnu riječ `abstract` kao modifikator prilikom definiranja klase.
- Apstraktne klase mogu imati konkretne i apstraktne metode:
  - Apstraktni metodi kreiraju se pomoću ključne riječi `abstract` u ulozi modifikatora metoda i nemaju implementacije.
  - Konkretni metodi imaju implementaciju i za njih važe pravila kao za obične klase.
- Da bi klasa koja je izvedena od apstraktne postala konkretna mora implementirati sve apstraktne metode, u suprotnom mora biti i sama označena kao apstraktna.

Primjer Java apstraktna klasa


```
abstract class GrafObjekat {
    private int x_, y_;
    public GrafObjekat(int x, int y) {
       x_ = x;
       y_ = y;
    }
    public abstract void crtaj();
    public final void pomjeri(int dx, int dy) {
       x_ += dx;
       y_ += dy;
    }
}
class Krug extends GrafObjekat {
    private int r_;
    public Krug(int r) {
       super(0,0);
       r_ = r;
    }
    public void crtaj() {
       System.out.println("Crtam krug r=" + r_);
    }
    public static void main(String[] args) {
       // GrafObjekat a = new GrafObjekat(2,3); generira gresku
       GrafObjekat b = new Krug(10);
       b.pomjeri(5,10);
       b.crtaj();
    }
}
```

Interface

- Java interface je sličan C++ klasi koja ne definira članove i ima samo čiste virtuelne metode.
- Kreiranje interface-a slično je kreiranju klase samo se koristi ključna riječ `interface` umjesto `class`
  - Nije moguće kreirati objekat od interface-a, ali je moguće definirati referencu na interface
  - Svi metodi koji pripadaju interface-u po default-u su javni i ne implementiraju se unutar interface-a
  - Klasa koja implementira interface mora obezbijediti implementaciju za svaki metod.
  - Klasa može da implementira proizvoljan broj interface-a.
    - `class Klasa1 extends Bazna implements Obojiv, Pomjerljiv {}`
  - Interface ne može da sadrži članove ali može da definira konstante
  - Interface može da nasljeđuje od drugih interface-a pomoću ključne riječi `extends`

Još o interface-ima

- Opšti oblik kreiranja interface-a

```
modifikator interface ImeInterfacea
{
        deklaracija metoda;
        deklaracija konstanti;

}
```

- Da bi mogli sortirati niz sa objektima neke klase (pomoću metoda `sort` iz klase `Arrays`), klasa mora implementirati interface `Comparable`, koji ima sljedeću definiciju:

```
public interface Comparable
{
        int compareTo(Object other);
}
```

- Pri čemu `compareTo` metod treba da vrati
  - Negativan broj ako objekat `this` "manji" od `other`
  - 0 ako su `this` i `other` "isti“
  - Pozitivan ako je objekat `this` "veći" od `other`

Primjer Java interface


```
import java.util.*;
class Radnik implements Comparable {
    private String ime_;
    private double plata_;
    Radnik(String ime, double plata) {
       ime_ = ime;
       plata_ = plata;
    }
    public int compareTo(Object other) {
       Radnik r = (Radnik) other;
       return ime_.compareTo(r.ime_);
    }
    public String toString() {
       return "ime=" + ime_ + " plata=" + plata_;
    }

    public static void main(String[] args) {
       Radnik[] niz = new Radnik[3];
       niz[0] = new Radnik("Deni",12.3);
       niz[1] = new Radnik("Dino",24);
       niz[2] = new Radnik("Damir",234);
       Arrays.sort(niz);
       if (niz[0] instanceof Comparable)
           System.out.println("Radnik je Comparable");
       for (int i = 0; i < niz.length; ++i)
           System.out.println(niz[i]);
    }
}
```

Primjer exception


```
class Bug1
{
   public static void main(String[] args)
   {
     int[] a = new int[3];
     int b = a[5];
   }
}
class Bug2
{
   public int clan = 10;
   public static void main(String[] args)
   {
     Bug2[] obj = new Bug2[3];
     int c = obj[1].clan;
   }
}
```

Exception (iznimka)

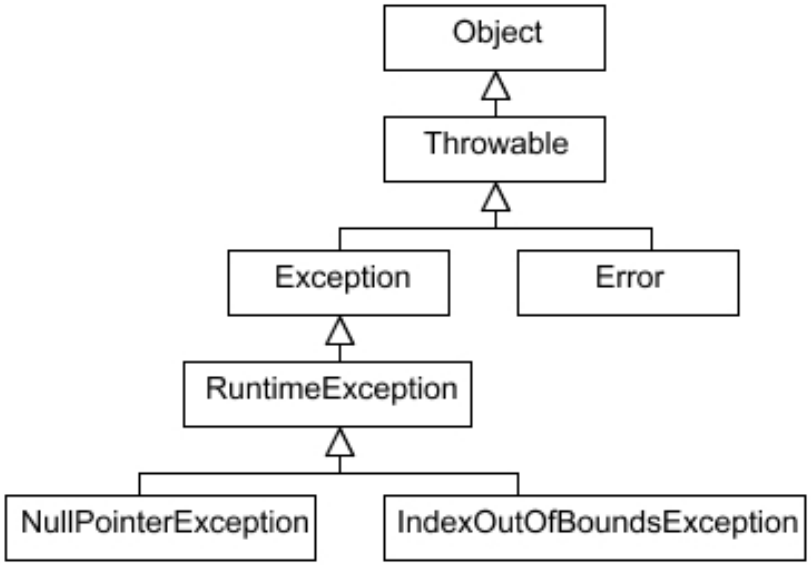
- Rezultati izvršenja programa:

```
$ java Bug1
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:5
   at Bug1.main(Test1.java:6)
$ java Bug2
Exception in thread "main" java.lang.NullPointerException at
   Bug2.main(Test1.java:15)
```

- Java programi bez obzira na programerske pogreške nikada ne bi trebali da naprasno terminiraju izvođenje (core dump), već generiraju iznimku koju je moguće tretirati u programu.
- U primjeru indeksiranje izvan granica niza generira `ArrayIndexOutofBoundsException` a dereferenciranje null reference generira `NullPointerException` iznimku.

Standardne iznimke

- Generirana iznimka je i sama je objekat tipa neke klase
- Standardna biblioteka definira veliki broj klasa koje se koriste za generiranje iznimki za različita abnormalna stanja u kojim se mogu naći programi.
- Standardne iznimke su organizirane u nasljednoj hijerarhiji (na slici je samo mali segment hijerahije):



Tretiranje iznimke

- Netretirane iznimke
  - Ako se u kodu eksplicitno ne tretira iznimka a ona nastane tokom izvršenja, JVM prekida sa izvođenjem programa i ispisuje se StackTrace (poruka o iznimci skupa sa sekvecom poziva funkcija koji je doveo do generiranja iznimke).
  - Da bi se ovo izbjeglo, iznimku je moguće eksplicitno tretirati pomoću `try` i `catch` blokova
- Ključna riječ `try`
  - Ukoliko u nekom dijelu koda može nastati iznimka, taj se dio koda ubacuje u poseban blok koji započinje sa instrukcijom try čime je naglašeno da će se eventualne iznimke u tom bloku biti tretirane.
  - Na mjestu gdje iznimka nastane, blok `try` momentalno prestaje sa izvršenjem i prelazi se na tretman generirane iznimke u nekom od `catch` blokova.

Tretiranje iznimke

- Generalni format try-catch bloka

```
try {
        //... neki kod
}
catch (tip_iznimke1 obj) {
        // tretiraj iznmku tipa tip_iznimke1
}
catch (tip_iznimke2 obj) {
        // tretiraj iznimku tipa tip_iznimke2
}
        ...
finally {
        // odradi nesto
}
```

- Objekat iznimka, u ovisnosti od svog tipa, proslijedit će se u odgovarajući `catch` blok na tretman
- Program nastavlja sa izvršenjem nakon tretmana u `catch` bloku
- Ako postoji, `finally` blok se izvršava u svakom slučaju, bilo da se izrši blok `try` ili bilo koji od `catch` blokova

Primjer2 exception


```
class Bug3
{
    public static void parsiraj(int[] ulaz,String[] args)
    {
       for (int i=0; i < args.length; ++i)
           ulaz[i] = Integer.parseInt(args[i]);
       System.out.println("Konverzija uspjela");
    }
    public static void main(String[] args)
    {
       int[] niz = new int[5];
       try
       {
           parsiraj(niz,args);
       }
       catch (NumberFormatException e)
       {
           System.out.println("Greska u unosu");
           System.out.println(e);
       }
       catch (RuntimeException e)
       {
           System.out.println("Greska u indeksiranju");
           e.printStackTrace();
       }
       System.out.println("Program nastavlja dalje");
    }
}
```

Primjer izvođenje


```
$ java Bug3 2 3 4 2 1 10
Greska u indeksiranju
java.lang.ArrayIndexOutOfBoundsException: 5
        at Bug3.parsiraj(Test2.java:6)
        at Bug3.main(Test2.java:14)
Program nastavlja dalje

$ java Bug3 2 3 k 3 2 1 1
Greska u unosu
java.lang.NumberFormatException: For input string: "k"
Program nastavlja dalje

$ java Bug3 2 3 4 2
Konverzija uspjela
Program nastavlja dalje
```

- Generalno
  - Izvršit će se onaj `catch` blok čiji tip odgovara tipu generirane iznimke
  - Ako je tip generirane iznimke izvedena klasa od klase koja je deklarirana u nekom `catch` bloku onda taj blok na sebe može preuzeti odgovornost tretiranja greške.

Automatsko zatvaranje resursa

- Postoji “try-with-resources” naredba koja automatski zatvara resurse
- U sljedećim naredbama fajl će se zatvoriti ukoliko se ili try ili catch blok izvrši

```
System.out.println("Otvaranje fajla");
try(InputStream in = newFileInputStream("missingfile.txt")) {
  System.out.println(“Fajl je otvoren");
  int data = in.read();
} catch(FileNotFoundException e) {
  System.out.println(e.getMessage());
} catch(IOException e) {
  System.out.println(e.getMessage());
}
```

Catch blok sa više tipova grešaka

- Postoji multi-catch izraz koji omogućava obradu više tipova grašaka u istom catch bloku

```
ShoppingCart cart = null;
try(InputStream is = newFileInputStream(cartFile);
    ObjectInputStream in = new ObjectInputStream(is)) {
        cart = (ShoppingCart)in.readObject();
} catch(ClassNotFoundException | IOException e) {
  System.out.println("Exception deserializing“ + cartFile);
  System.out.println(e);
  System.exit(-1);
}
```

Generiranje iznimke

- Bilo koji metod može generirati iznimku
  - Za generiranje iznimke koristi se ključna riječ throw praćeno sa objektom koji se generira kao iznimka npr.
    - `throw new IOException("Dogodila se greska");`
- Ako se unutar nekog metoda generira iznimka ili više njih, a da se iste ne tretiraju unutar bloka tog metoda, onda se taj metod, nakon liste parametara, **mora označiti** sa riječi `throws` praćenom sa listom koja deklariše sve iznimke koje se generišu a nisu tretirane unutar metoda (ovo važi za tzv checked exceptions) npr
  - `public int ucitaj(string par) throws IOException`
  - `{ throw new IOException("Dogodila se greska");}`
- Dva tipa iznimki (tzv unchecked exceptions) ne moraju se nabrajati u listi netretiranih iznimki i to:
  - iznimke tipa `RuntimeException` i sve njene subklase
  - iznimke tipa `Error` (koje se obično i ne tretiraju)

Primjer


```
class Test {
    public static int podijeli(int a, int b) throws Exception {
      if(b == 0)
          throw new Exception("Dijeljenje sa 0");
      return a/b;
    }
    public static int pozovi2(int a, int b) throws Exception {
      return podijeli(a,b);
    }
    public static void pozovi1(int a, int b) {
      int temp;
      try
      {
          temp = pozovi2(a,b);
      }
      catch(Exception e)
      {
          System.out.println("Uhvaceno: " + e + " Generisem novu gresku");
          //throw e; greska u kompajliranju
          throw new RuntimeException("Dogodila se greska");
      }
    }
    public static void main(String[] args) {
      int a = 2, b = 0;
      pozovi1(a,b);
    }
}
```

Kreiranje korisničke iznimke


```
$ java Test
Uhvaceno: java.lang.Exception: Dijeljenje sa 0 Generisem novu gresku
Exception in thread "main" java.lang.RuntimeException: Dogodila se greska
        at Test.pozovi1(Test3.java:24)
        at Test.main(Test3.java:30)
```

- Korisničke iznimke moraju biti tipa koji nasljeđuje od `throwable`
  - Za kreiranje checked iznimki koristiti `Exception` kao baznu klasu
  - Za kreiranje unchecked iznimki koristiti `RuntimeException` kao baznu klasu.
- Primjer:

```
public class MojException extends Exception
{
    MojException(String poruka)
    {
        super("MojException generiran: " + poruka);
    }
}
```

Assertions (tvrdnje)

- Forma testiranja koja omogućava provjeru tačnih pretpostavki u kodu
- Assertions se mogu onemogućiti u runtime-u
- Sintaksa:

```
assert<boolean_expression> ;
assert<boolean_expression> : <detail_expression> ;
```

  - Ako se `boolean_expression` evaluira kao `false` izbacuje se `AssertionError` iznimka
  - Drugi argument je opcioni i koristi se za opis izbačene iznimke