OBJEKTNO-ORIJENTISANI JEZICI

Apstrakcija tipova podataka

Klase kao sredstvo apstrakcije

- Enkapsulacija
- Nasleđivanje
- Polimorfizam

Generički tipovi

Pojam apstrakcije

- Konceptualno pojednostavljenje prikaza različitih entiteta pri čemu se namerno ignorišu detalji a ističu globalna svojstva

- Koncept apstrakcije je fundamentalan u programiranju
  - Apstrakcija procesa
    - Korišćenjem potprograma – potprogram definiše neki proces, a njegov poziv je apstrakcija tog procesa, korisnik potprograma nije upoznat sa detaljima implementacije
  - Apstrakcija podataka

Apstrakcija podataka

- Apstraktni tipovi podataka su korisnički definisani tipovi koji zadovoljavaju sledeće uslove:
  - Reprezentacija podataka tog tipa je sakrivena od delova programa koji te podatke koriste. Nad podacima tog tipa mogu da se primenjuju samo operacije definisane u tipu.
  - Deklaracija podataka i operacija koje su nad njima primenljive se definišu u jedinstvenom modulu, drugi programski moduli mogu samo da kreiraju promenljive definisanog tipa. (**Enkapsulacija**)

Sredstva za apstrakciju podataka

- U proceduralnim jezicima:
  - Moduli
  - Jezici koji podržavaju rad sa modulima:
    - Pascal,
    - Modula,
    - Ada (objektno-orijentisani jezik koji podržava i definiciju modula)
- U objektno orijentisanim jezicima
  - Klase

Moduli kao sredstva za apstrakciju podataka

- Modul sadrži:
  - Interfejs – sadrži definicije tipova podataka koji se u modulu obradjuju i deklaracije potprograma koje podatke tih tipova obradjuju
  - Implementaciju – sadrži implementaciju potprograma definisanih u interfejsu modula

Moduli u Paskalu

```
UNIT StackLib;

INTERFACE
CONST Height = 30; {  maximum size }
TYPE RANGE =  1..Height;
ItemType = CHAR;
StackType = RECORD
Top : RANGE;
Item: ARRAY[RANGE] OF  ItemType;
END;
PROCEDURE Create(VAR  Stack:StackType );
PROCEDURE Push(VAR Stack:StackType; X:ItemType);
PROCEDURE Pop(VAR Stack:StackType; VAR Y:ItemType);
FUNCTION Empty(Stack:StackType): BOOLEAN;
FUNCTION Full(Stack:StackType ): BOOLEAN;

IMPLEMENTATION
PROCEDURE Create(VAR  Stack: StackType);
BEGIN
  Stack.Top := Height;
END; { Create }
FUNCTION Empty(Stack:  StackType): BOOLEAN;
BEGIN
  IF Stack.Top = Height  THEN
    Empty := TRUE
  ELSE
  Empty := FALSE;
END; { Empty }
```

```
FUNCTION Full(Stack:StackType): BOOLEAN;
BEGIN
  IF Stack.Top = 1 THEN
    Full := TRUE
  ELSE
    Full := FALSE;
END; { Full }
PROCEDURE Push(VAR Stack:StackType; X:ItemType);
BEGIN
  IF Full(Stack) THEN
    WriteLn('FULL ')
  ELSE BEGIN
    DEC(Stack.Top);
    Stack.Item[Stack.Top]  := X;
  END;
END; { Push }
PROCEDURE Pop(VAR Stack:StackType; VAR Y:ItemType);
BEGIN
  IF Empty(Stack) THEN
    WriteLn('EMPTY ')
  ELSE BEGIN
    Y := Stack.Item[Stack.Top];
    Inc(Stack.Top);
  END;
END; { Pop }
END. { StackLib }
```

Koncept objektno-orijentisanog programiranja

- Svi podaci koji se u programu obradjuju predstavljaju objekte.
- Objekat je definisan podacima koji opisuju njegovo stanje i skupom metoda koji opisuju njegovo ponašanje.
- Objekti komuniciraju međusobno slanjem poruka (pozivanjem metoda).
- Svaki objekat pripada svojoj klasi (ima svoj tip).

Klase kao sredstvo za apstrakciju podataka

- Enkapsulacija
  - Klasa sadrži skup podataka koji opisuju stanje objekta i metode (operacije) koje se nad objektom mogu izvršavati.
- Sakrivanje podataka
  - Za sve članove klase je definisano pravo pristupa:
    - **Public**
    - **Protected**
    - **Private**
  - Nepisano pravilo: **svi podaci treba da budu privatni**.

Klase u programskom jeziku C++

- Definicija klase odvojena od implementacije.
- Pravima pristupa se postiže sakrivanje podataka.

```
class <ImeKlase>
{
  public:
    <niz_javnih_clanova>
  protected:
    <niz_zasticenih_clanova> ]
  private:
    <niz_privatnih_clanova>
};
```

Interfejs

Nedostupan deo

Primer klase u programskom jeziku C++

```
// Stack.h - the header file for the Stack class
#include <iostream.h>
class Stack {
private: //** These members are visible only to other
//** members and friends (see Section 11.6.4)
  int *stackPtr;
  int maxLen;
  int topPtr;
public: //** These members are visible to clients
  Stack(); //** A constructor
  ~Stack(); //** A destructor
  void push(int);
  int pop();
  int top();
  bool empty();
}
```

Primer klase u programskom jeziku C++

```
// Stack.cpp - the implementation file for the Stack class
#include <iostream.h>
#include "Stack.h"
using std::cout;
Stack::Stack() { //** A constructor
  stackPtr = new int [100];
  maxLen = 99;
  topPtr = -1;
}
Stack::~Stack() {delete [] stackPtr;}; //** A destructor
void Stack::push(int number) {
  if (topPtr == maxLen)
  cerr << "Error in push--stack is full\n";
  else stackPtr[++topPtr] = number;
}
...
```

Klase u programskom jeziku Java

- Implementacija funkcija direktno u klasi.
- Dodatno (podrazumevano) pravo pristupa – pravo pristupa na nivou paketa
  - Sve definisano u okviru jednog paketa se smatra „prijateljima“
  - Sve definisano u okviru jednog paketa čemu nije eksplicitno dodeljeno pravo pristupa se može koristiti u svim funkcijama članicama svih klasa definisanih u okviru istog paketa.
- Enkapsulacija u Javi:
  - Enkapsulacija na nivou klase slabija nego u C++-u
  - Enkapsulacija na nivou paketa

Primer klase u programskom jeziku Java

```
class Stack {

        private int [] *stackRef;
        private int maxLen, topIndex;
        public StackClass() { // a constructor
                stackRef = new int [100];
                maxLen = 99;
                topPtr = -1;
        }
        public void push (int num) {…}
        public int pop () {…}
        public int top () {…}
        public boolean empty () {…}
        public boolean full () {…}
}
```

Klase u programskom jeziku C#

- Implementacija funkcija direktno u klasi.
- Dodata svojstva kao posebni članovi klase
  - Spolja se vide kao javni atributi
  - Unutra su to funkcije (get i/ili set)
- Dodatna prava pristupa:
  - **Internal** – članovi vidljivi u okviru istog asemblija (**assembly**), ovo je podrazumevano pravo pristupa
  - **Protected internal** – članovi vidljivi u okviru istog asemblija i u izvedenim klasama
- Enkapsulacija na nivou asemblija

Pojam asemblija u C#-u

- Deo aplikacije koji sarži semanatički povezan skup klasa koji se nezavisno prevodi.
- Svakom asembliju odgovara poseban projekat.
- Na osnovu jednog asemblija generiše se:
  - Modul koji se može izvršavati (.exe), ili
  - Biblioteka klasa u majkrosoftovom medjukodu Common Intermediate Language (CIL) – Dynamic link library (.dll)
- Klase unutar jednog asemblija se tretiraju kao „prijateljske“
  - Sve funkcije članice svih klasa definisanih u okviru jednog asemblija mogu pristupati članovima svih klasa istog asemblija za koje nije eksplicitno navedeno pravo pristupa, ili je navedeno pravo pristupa `internal`.

Novine koje OO paradigma donosi

- Nasleđivanje
- Polimorfizam

Nasleđivanje

- Nasleđivanje omogućava da se definišu nove klase na osnovu postojećih.
- Nova klasa se definiše kao specijalizacija ili proširenje neke klase koja već postoji.
- Nova klasa inicijalno ima sve atribute i metode klase koju nasleđuje.
- Novoj klasi mogu da budu pridodati novi atributi i nove metode.
- Metode mogu biti izmenjene sa ciljem da se nova klasa prilagodi specifičnim potrebama.

Definicija izvedene klase

- C++:

  `class <ClassName> : <Parent1>[, <Parent2>]`…

  gde je definicija roditeljske (<ParentK>):

  `access-specifier`$_{\texttt{optional}}$ **`virtual`**$_{\texttt{optional}}$ **`<ParentClassName>`**

- Java:

  `class <ClassName> extends <ParentClassName>`

- C#:

  `class <ClassName> : <ParentClassName>`

Načini izvođenja

C++

- Public
  - Objekat izvedene klase JE I objekat roditeljske klase (IS A)
- Protected
- Private
  - Objekat roditeljske klase JE DEO objekat roditeljske klase (PART OF)

Java i C#

- Samo jedan način izvođenja koji odgovara javnom izvođenju u C++-u.
- Kada je potrebno da objekat jedne klase bude deo druge, definiše se kao atribut.

Polimorfizam

- Termin označava “više formi” - "many forms", i u kontekstu objektnih jezika ukazuje na mogućnost da se različiti metodi pozivaju preko istog interfejsa.

- Sposobnost da raziličiti objekti odgovore ne iste poruke na sebi svojstven način.

Polimorfizam u jezicima C++, Java i C#

Obeležavanje polimorfnih funkcija:

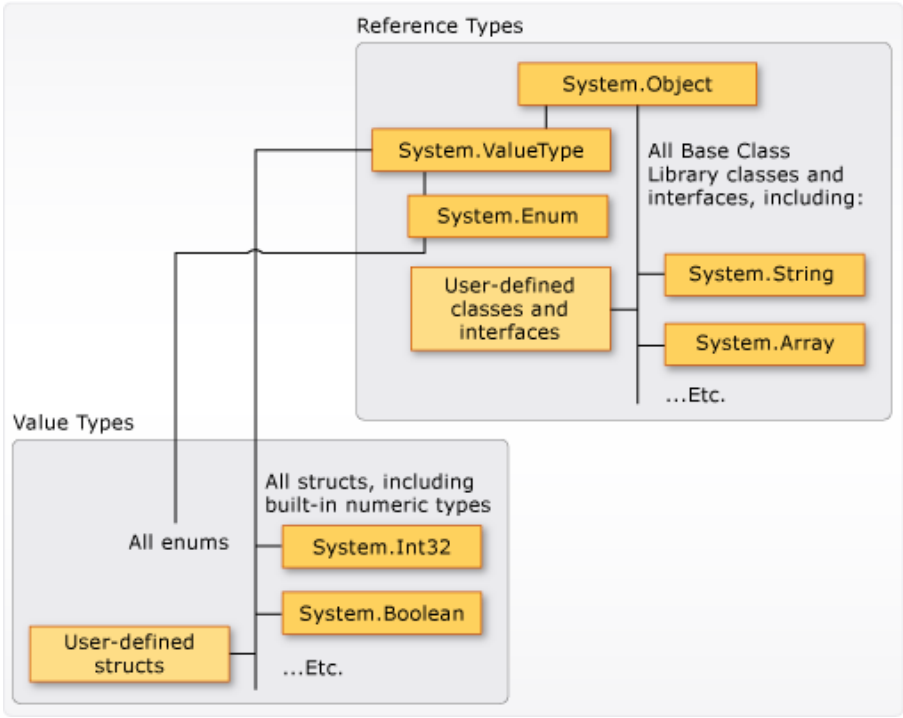
	C++	Java	C#
U osnovnoj klase	`virtual`	-	`virtual`
U izvedenoj	-	-	`override`

Aktiviranje polimorfizma pri pozivu funkcije:

C++			Java	C#
Objekat	Pokazivač	Referenca	Referenca	Referenca
-	+	+	+	+

Tip `Object` u programskim jezicima Java i C#

- Java:
  - Iz klase Object su izvedene sve klase
  - Ako se definiše klasa bez roditeljske klase, biće direktno izvedena iz klase Object
- C#:
  - Iz klase Object izvedeni svi tipovi podataka



Apstraktne klase

- Sadrže bar jednu apstaktnu metodu
  - Apstraktna metoda nema implementaciju
- Ne mogu se kreirati objekti apstraktnih klasa
- Služe kako kalup za pravljenje novih klasa

Apstraktne klase u jezicima C++, Java i C#

Obeležavanje:

	C++	Java	C#
Apstraktne klase	-	`abstract`	`abstract`
Apstraktne metode	`virtual ... =0`	`abstract`	`abstract`
Metode koje predefinišu apstraktne	-	-	`override`

Interfejsi

- Definišu skup apstraktnih metoda koje izvedene klase treba da implementiraju.
- Metode članice interfejsa su uvek apstraktne i javne.
- U Javi: članovi interfejsa mogu da budu atributi koji su obavezno javni, statički i konstantni.
- U C#-u: članovi interfejsa mogu biti i sojstva (properties), delegati (delegates) i događaji (events). Svi oni su apstraktni i javni.

Interfejsi u jezicima Java i C#

- Definicija interfejsa:

```
interface <ImeInterfejsa> { … }
```

- Navođenje modifikatora public i abstract ispred metoda:
  - U Javi: može ali ne mora.
  - U C#-u: NE.

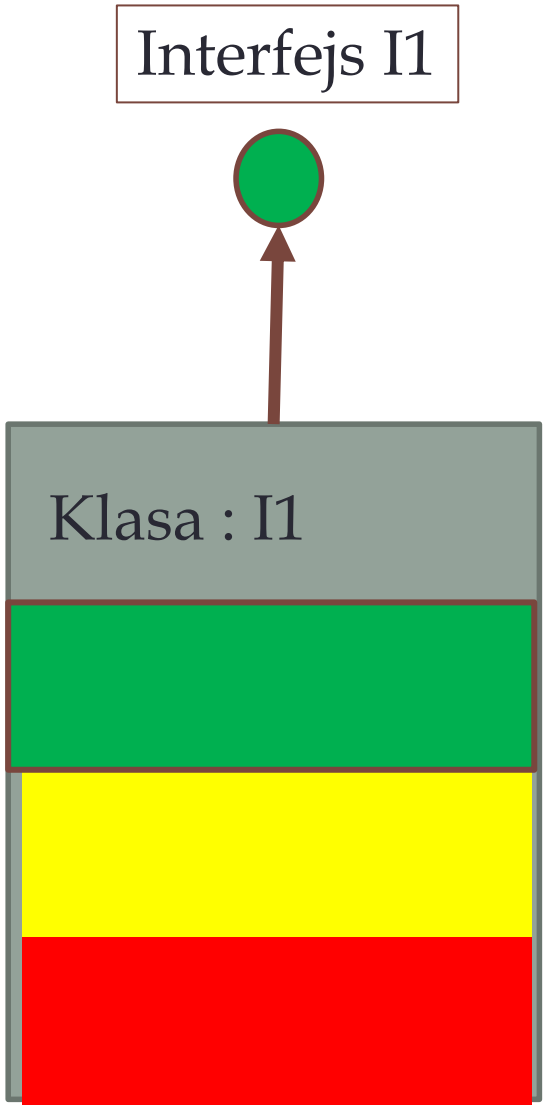
- Definicija klase koja implementira interfejs:
  - U C#-u:

```
class <ImeKlase> : <ImeInterfejsa> { … }
```

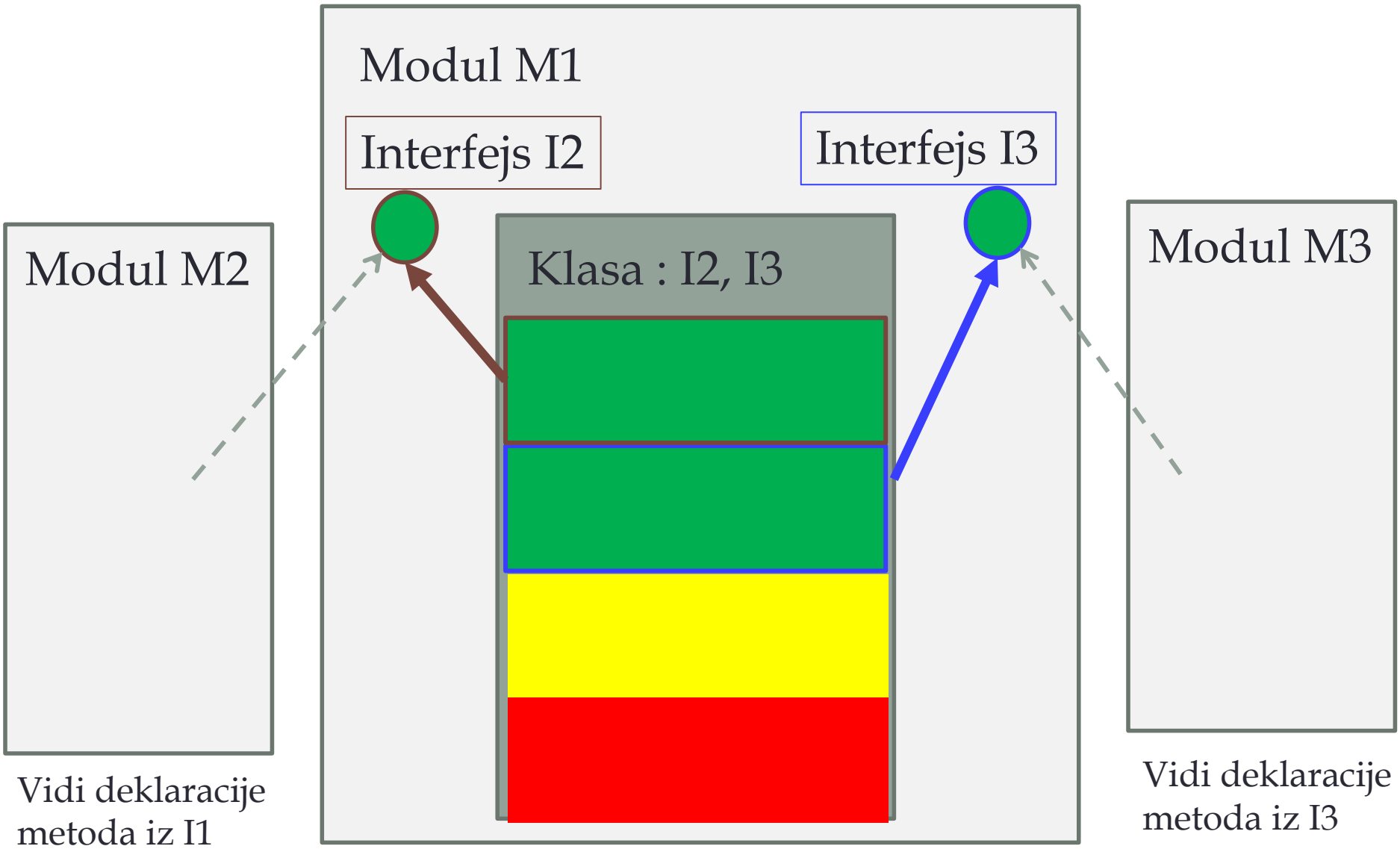
  - U Javi:

```
class <ImeKlase> implements <ImeInterfejsa> { … }
```

Interfejsi kao sredstvo za enkapsulaciju



Interfejsi kao sredstvo za enkapsulaciju



Generičke funkcije, generički tipovi

- Funkcije ili klase koje realizuju često korišćene algoritme ili strukture podataka sa nepoznatim tipovima podataka (a u C++-u i sa nepoznatim vrednostima nekih konstanti)

- Templejti, templejtske funkcije i klase (C++ notacija)
- Generičke funkcije i klase (Java i C# notacija)
- Parametrizovani tipovi

Templejti u programskom jeziku C++

```
template <ParametriTemplejta>
DefinicijaFunkcijeIliKlase
```

- Parametar templejta može biti:
  - Tip: `class/typename ImeNepoznatogTipa`, ili
  - Constanta: `Tip ImeNepoznateKonstante`

- Primer:

```
template <class T, int n>
void sort(T a[n]) { … }
```

Primer templejtske klase u C++-u

- Klasa za predstavljanje magacina sa nepoznatim tipom elemenata:

```
template <class Type>
class Stack
{
  private:
       Type *stackPtr;
       const int maxLen;
       int topPtr;
  public:
       Stack() {  // Constructor for 100 elements
          stackPtr = new Type[100];
          maxLen = 99;
          topPtr = -1;
       }
       Stack(int size) {  // Constructor for a given number
          stackPtr = new Type[size];
          maxLen = size - 1;
          topPtr = -1;
       }
       ...
}
```

- Kreiranje instance: `Stack<int> myIntStack;`

Generičke funkcije i klase u Javi

- Prvi put uvedene u verziji Java 5.0
  - Smatralo se da tip Object može da zameni nepoznati tip
    - Problem: stalno kastovanje
- Parametar templejta može biti samo tip
- Na mesto stvarnog parametra templejta se ne mogu koristiti primitivni tipovi
  - Koriste se wrapper klase
- Sve bibliotečke klase za predstavljanje kolekcija su zamenjene generičkim klasama:
  - `ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet`

Generičke funkcije i klase u Javi

- Definicija generičke funkcije

```
pravo_pristupa <argumenti_templejta>
tip ime(argumenti_funkicije)
{
    ...
}
```

- Definicija generičke klase

```
class Ime<argumenti_teplejta>
{
    ...
}
```

Upotreba parametara templejta

Paramerar templejta se **može** iskoristiti kao:

- Tip atributa,
- Tip parametra metode,
- Tip lokalne promenljive metode,
- Povratni tip metode.

Parametar templejta se **ne može** iskoristiti kao:

- Tip statičkog atributa,
- Tip parametra statičke metode,
- Tip lokalne promenljive u statičkoj metodi,
- Povratni tip statučke metode,
- Tip niza ili objekta koji se kreira.

Generička klasa u Javi - primer

```
class Stack<T> {
   public void push(T newElement){…}
   public T pop();
   ...
}
```

Instanciranje objekata tipa templejtske klase:

```
   Stack<Integer> intStack = new
   Stack<Integer>();
   Stack<String> strStack = new
      Stack<String>();
```

Ili

```
   Stack<Integer> intStack = new Stack< >();
   Stack<String> strStack = new Stack< >();
```

Dodavanje ograničenja parametrima templejta

- U definiciji templejta mogu se dodavati ograničenja koja treba da zadovolje tipovi da bi se mogli upotrebiti kao argument templejta pri instanciranju objekta.
- Definicija ograničenja:

```
T extends B1 & I1 & I2 & ... & In
```

Primer generičke klase sa ograničenim parametrima

```
class Vektor<T extends Comparable>
{
   private T[] v;
   ...
   public T min()
   {
      T m = v[0];
      for (int i=0; i<v.Length; i++ )
        if ( v[i].CompareTo(m) > 0 )
            m = v[i];
      return m;
   }
}
```

Upotreba džokera na mesto stvarnih parametara generičkih klasa

- Na mesto stvarnog argumenta templejta se može upotrebiti
  - Neograničeni džoker: `?`
  - Ograničeni džoker: `? extends B, ? super B`

- Primer korišćenje džokera na mesto stvarnog argumenta:

```
Stack<?>  s = new Stack<Integer>();
...
 s = new Stack<String>();
```

Generičke funkcije i klase u C#-u

- Uvedene u verziji C# 2005.
- Parametar templejta može biti samo tip
- Na mesto stvarnog parametra templejta se mogu koristiti primitivni tipovi
- U biblioteci klasa za predstavljanje kolekcija paralelno definisane generičke i negeneričke klase (čiji su elementi tipa Object)
  - Negeneričke kolekcije:
    - `ArrayList, SortedList, Stack, Queue, Hashtable`
  - Generičke kolekcije:
    - `List, SortedList, Dictionary, Stack, Queue, Hashset`

Definicija generičke klase u C#-u

```
class Ime<argumenti_teplejta> [<ogranicenja_argumenata>]
{
    ...
};
```

Gde se u ograničenjima parametara pišu uslovi koje treba da zadovolje tipovi koji se koriste kao stvarni argumenti templejta.

Definicija ograničenja:

```
where <argument_templejta> : <ogranicenje>
```

Vrste ograničenja argumenata templejta

- **`class`** - argument templejta je referentnog tipa
- **`struct`** – argument templejta je vrednosnog tipa
- **`new()`** - argument templejta je klasa koja sadrži javni konstruktor bez argumenata
- **`<ime_klase>`** - argument templejta je tipa date klase ili klase direktno ili indirektno izvedene iz date klase
- **`<ime_interfejsa>`** - argument templejta može biti samo klasa koja implementira dati interfejs

Primer generičke klase sa ograničenim argumentima

```
class Vektor<T> where T : IComparable
{
    private T[] v;
    public Vektor(int n) { v = new T[n]; }
    public T min()
    {
        T m=v[0];
        for (int i=0; i<v.Length; i++ )
          if ( ((IComparable)v[i]).CompareTo(m) > 0 )
              m=v[i];
        return m;
    }
}
```

Generičke metode u C#-u

- U C#-u se može definisati genirčka metoda u klasi koja nije generička.
- Definicija generičke metode:

```
tip imef<argumenti_templejta>(...) { … }
```

- Poziv generičke metode:

```
imef<lista_tipova>(…)
```

Razlika između generičkih klasa i klasa sa atributima tipa Object

```
class Stack<T>
{
   private T [] v;
   private int k=0;

   public Stack(int n) { v =
    new T[n]; }
   public void Push(T novi)
    {
      v[k++] = novi;
    }
   public T Pop()
   {
     return v[--k];
   }
}
```

```
class Sack
{
  private Object [] v;
  private int k=0;

  public Stack(int n) { v = new
    Object[n]; }
  public void Push(Object novi)
  {
     v[k++] = novi;
  }
  public Object Pop()
  {
     return v[--k];
  }
}
```

Razlika između generičkih klasa i klasa sa atributima tipa Object

```
Stack<int> intStack;
intStack = new
   Stack<int>(30);
int k=5;
intStack.push(k);
int k2=s.pop();
```

```
Stack s;
s = new Stack();
int k=5;
S.push(k);
int k2 = (int)
   s.pop();
float z=10;
s.push(z);
```

Razlika između generičkih klasa i klasa sa atributima tipa Object

- Generička klasa
  - Ako se radi o jednoj kolekciji, svi članovi kolekcije su istog tipa
  - Uvodjenjem ograničenja sužava se lista tipova koji se mogu koristiti kao argumenti templejta tako da korišćenje nekih operacija može biti bezbedno
  - Korišćenje atributa van klase se vrsi bez eksplicitnog kastovanja
- Klasa sa atributima tipa Object
  - Članovi kolekcije mogu biti raznorodni
  - Ne postoje nikakva ograničenja nad tipovima kojima mogu pripadati odgovarajuci atributi
  - Ukoliko se atributi koriste spolja, moraju se stalno kastovati u svoj pravi tip