Универсальность. Классы с родовыми параметрами

Под универсальностью (genericity) понимается способность класса объявлять используемые им типы как параметры.

Класс с параметрами, задающими типы, называется **универсальным классом (generic class)**.

Универсальными могут быть как классы, так и все их частные случаи - **интерфейсы, структуры, делегаты, события**.

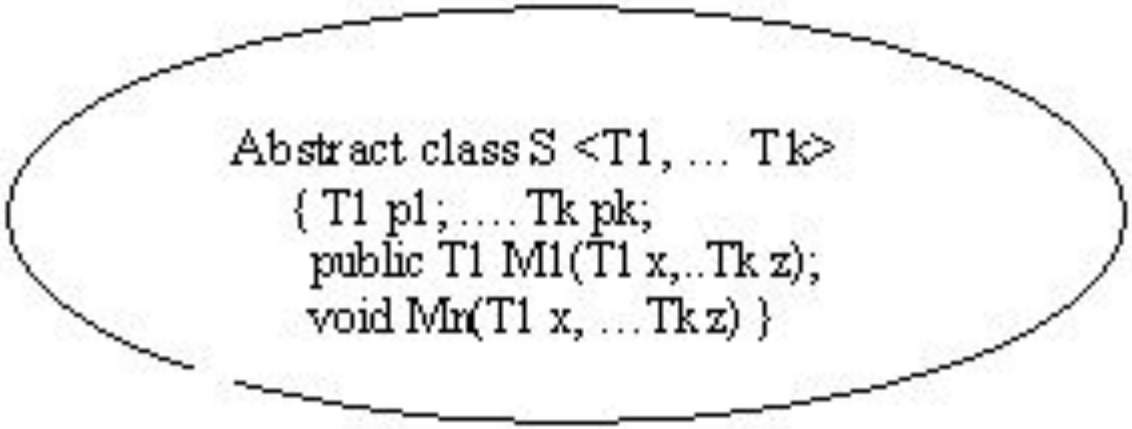
class MyClass<T1, ... Tn> {...}

Класс с универсальными методами

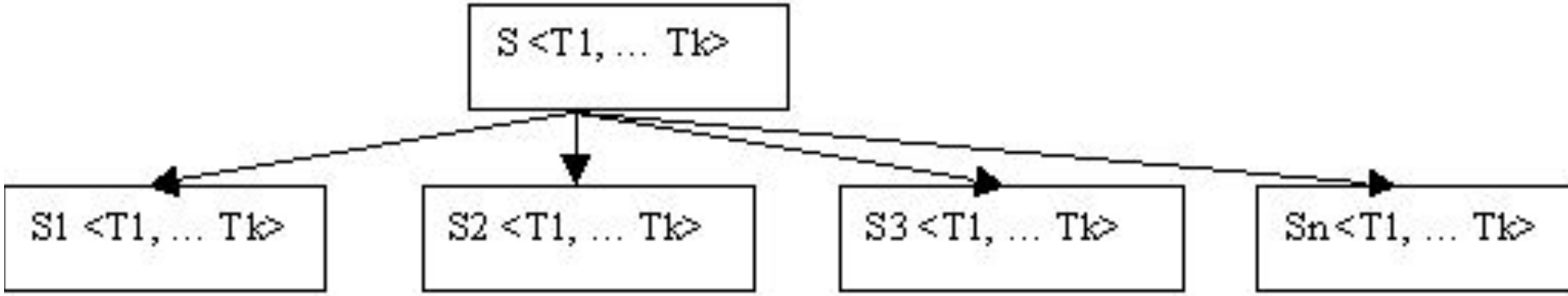
```
class Change{
   static public void Swap<T>(ref T x1, ref T x2)   {
      T temp;
      temp = x1; x1 = x2; x2 = temp;
   }
}

public void TestSwap(){
   int x1 = 5, x2 = 7;
   Change.Swap<int>(ref x1, ref x2);
   string s1 = "Саша", s2 = "Павел";
   Change.Swap<string>(ref s1, ref s2);
   Person pers1 = new Person("Савлов", 25, 1500);
   Person pers2 = new Person("Павлов", 35, 2100);
   Change.Swap<Person>(ref pers1, ref pers2);
}
```

Два основных механизма объектной технологии

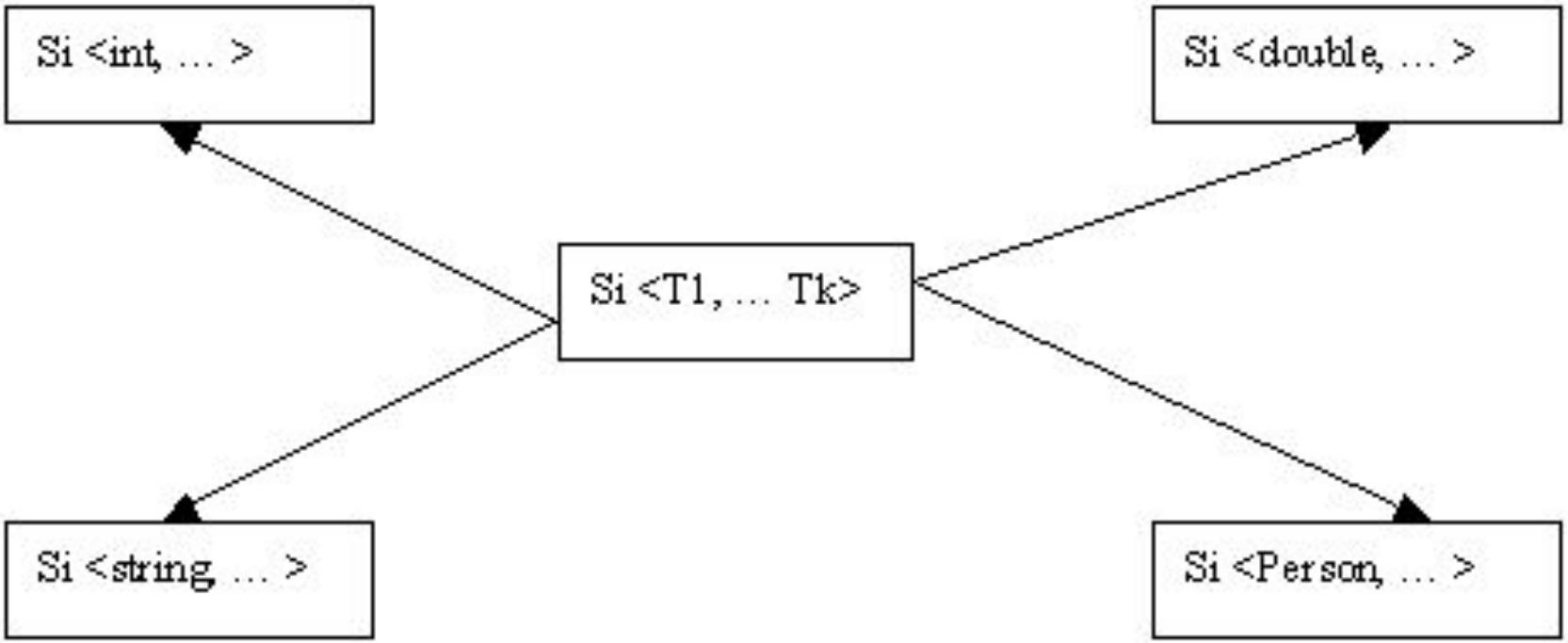
- Наследование позволяет специализировать операции класса, уточнить, как должны выполняться операции.
- Универсализация позволяет специализировать данные, уточнить, над какими данными выполняются операции.



Этап проектирования (спецификации):
абстрактный класс с абстрактными типами



Наследование: уточняется представление данных;
задается или уточняется реализация методов родителя



Родовое порождение: уточняются типы данных; порождается класс путем подстановки конкретных типов

Стек

```
abstract public class GenStack<T>{
   abstract public T item();
   abstract public void remove();
   abstract public void put(T t);
   abstract public bool empty();
}
```

Стек на односвязном списке

```
public class OneLinkStack<T> : GenStack<T>{
    public OneLinkStack()   {      top = null;   }
    GenLinkable<T> top ;    //ссылка на вершину стека
    public override T item()   {      return (top.Item);   }
    public override bool empty()   {      return (top == null);   }
    public override void put(T elem)   {
     GenLinkable<T> newitem = new GenLinkable<T>();
     newitem.Item = elem;
     newitem.Next = top;
      top = newitem;
    }
    public override void remove()   {top = top.Next;   }
}
public class GenLinkable<T>{
    public T Item;
    public GenLinkable<T> Next;
    public GenLinkable()   { Item = default(T); Next = null; }
}
```

Стек в виде массива

```
public class ArrayUpStack<T> : GenStack<T>{
    int SizeOfStack;
    T[] stack;
    int top;
    public ArrayUpStack(int size) {
     SizeOfStack = size;
     stack = new T[SizeOfStack];
     top = 0;
    }
    public override void put(T x)   { stack[top] = x; top++; }
    public override void remove()   { top--; }
    public override T item()   { return (stack[top-1]); }
    public override bool empty()   { return (top == 0); }
}
```

```
public void TestPerson()  {
    OneLinkStack<int> stack1 = new OneLinkStack<int>();
    OneLinkStack<string> stack2 = new OneLinkStack<string>();
    ArrayUpStack<double> stack3 = new ArrayUpStack  <double>(10);
    ArrayUpStack<Person> stack4 = new ArrayUpStack<Person>(7);
. . .
}
```

Ограниченная универсальность

- ***Ограничение наследования***. Это основный вид ограничений, указывающий, что тип T является наследником некоторого класса и /или интерфейсов. Следовательно, над объектами типа T можно **выполнять все операции, заданные базовым классом и интерфейсами**. **where T: BaseClass, I1, ...Ik.**

- ***Ограничение конструктора***. Это ограничение указывает, что тип T имеет конструктор без аргументов и, следовательно, позволяет создавать объекты типа T. **where T: new().**

- ***Ограничение value/reference***. Это ограничение указывает, к значимым или к ссылочным типам относится тип T. Для указания значимого типа задается слово **struct**, для ссылочных - **class**. **where T: struct**.

```
public class Father<T1, T2>
{ }
public class Base{
    public void M1() { }
    public void M2() { }
}
public class Child<T1,T2> : Father<T1,T2>
   where T1:Base, IEnumerable<T1>, new()
   where T2:struct, IComparable<T2>
{  }
```

Пример: список с возможностью поиска элементов по ключу

```
class Node<K, T> where K: IComparable<K> {
    public Node()   {
     next = null;
     key = default(K);
     item = default(T);
    }
    public K key;
    public T item;
    public Node<K, T> next;
}
```

```
public class OneLinkList<K, T> where K : IComparable<K> {
    Node<K, T> first, cursor;
    public void start(){ cursor = first; }
    public void finish(){
      while (cursor.next != null)
      cursor = cursor.next;
    }
    public void forth(){
      if (cursor.next != null) cursor = cursor.next;
    }
    public void add(K key, T item){
      Node<K, T> newnode = new Node<K, T>();
      newnode.key = key;
      newnode.item = item;
      if (first == null)   {
            first = cursor= newnode;
      }
      else   {
            newnode.next = cursor.next;
            cursor.next = newnode;
      }
    }
```

```
public bool findstart(K key){
    Node<K, T> temp = first;
    while (temp != null)   {
     if (temp.key.CompareTo(key) == 0) {
           cursor=temp;
           return(true);
     }
     temp= temp.next;
    }
    return (false);
}

public void TestConstraint(){

    OneLinkList<int, string> list1 = new OneLinkList  <int, string>();

    OneLinkList<string, Person> list2 = new OneLinkList < string, Person>();
    …
}
```

Родовое порождение класса.

- **using IntStack = Generic.OneLinkStack<int>;**

```
public void TestIntStack(){
    IntStack stack1 = new IntStack();
    IntStack stack2 = new IntStack();
    IntStack stack3 = new IntStack();
. . .
}
```

Частные виды классов

- *Универсальные структуры*

```
public struct Point<T>{
    T x, y;     //координаты точки
}
```

- *Универсальные интерфейсы*

```
IComparable<T>
```

- *Универсальные делегаты*

```
class WithDelegate<T>{
    public delegate T Del(T a, T b); - объявление делегата
    . . .
}
WithDelegate <int>.Del del1;
del1 = new WithDelegate <int>.Del(Program.max2);
либо del1= this.max2;
```

- *Функциональный тип-делегат с родовыми праметрами*

```
public delegate T FunTwoArg<T>(T a, T b);
```