UNIT-IV

Motivation for Generic Types

This chapter starts the lecture about *generics*: Generic types and generic methods. With *generics* we are aiming at more general types (classes, structs, interfaces, etc). The measure that we will bring into use is *type parametrization*.

This chapter is intended as motivation. Type parameterized types will be the topic of Chapter 42 and type parameterized methods will be treated in Chapter 43.

4.1. Operations on sets In this chapter we decide to develop and use the class Set. We use the class Set as a motivating example. It is our goal, once and for all, to be able to write a class Set that supports all possible types of elements. It is the intention that the class Set can be used in any future program, in which there is a need for sets.

It is noteworthy that .NET has not supported a mathematical set class until version 3.5. As of version 3.5, the class HashSet<T> supports sets, see also Section 4.1. Thus, at the time of writing this material, there was no set class available in the .NET Framework.

From a mathematical perspective, a set is an *unordered* collection of items *without duplicates*. The class Set represents a mathematical set of items. We equip class Set with the following well-known set operations:

- *aSet*.**Member**(*element*)
- *aSet*.**Insert**(*element*)
- *aSet*.**Delete**(*element*)
- *aSet*.**Count**
- *aSet*.**Subset**(*anotherSet*)
- *aSet*.**GetEnumerator**()
- *aSet*.**Intersection**(*anotherSet*)
- *aSet*.**Union**(*anotherSet*)
- *aSet*.**Diff**(*anotherSet*)

The set operations Intersection, Union, and Diff are handled in Exercise 11.1.

41.2. The classes IntSet and StringSet

Let us imagine that we first encounter a need for sets of integers. This causes us (maybe somewhat narrow-minded) to write a class called IntSet. Our version of class IntSet is shown in Program 41.1. The version provided in the paper version of the material is abbreviated to save some space. The version in the web version is complete with all details.

```
using System;
using System.Collections;

public class IntSet {

  private int capacity;
  private static int DefaultCapacity = 10;
  private int[] store;
  private int next;     // The next place to insert

  public IntSet(int capacity){
    this.capacity = capacity;
    store = new int[capacity];
    next = 0;
  }

  public IntSet(): this(DefaultCapacity){
  }

  public IntSet(int[] elements): this(elements.Length){
    foreach(int  el in elements) this.Insert(el);
  }

  // Copy constructor
  public IntSet(IntSet s): this(s.capacity){
    foreach(int  el in s) this.Insert(el);
  }

  public bool Member(int  element){
    for(int idx = 0; idx < next; idx++)
      if (element.Equals(store[idx]))
        return true;
    return false;
  }

  public void Insert(int  element){
    if (!this.Member(element)){
      if (this.Full){
        Console.WriteLine("[Resize to {0}]", capacity * 2);
        Array.Resize<int>(ref store, capacity * 2);
        capacity = capacity * 2;
      }
      store[next] = element;
      next++;
    }
  }

  public void Delete(int  element){
    bool found = false;
    int foundIdx = 0;
    for(int idx = 0; !found && (idx < next); idx++){
      if (element.Equals(store[idx])){
        found = true;
        foundIdx = idx;
      }
    }
    if (found){   // shift remaining elements left
      for(int idx = foundIdx+1; idx < next; idx++)
        store[idx-1] = store[idx];
      store[next-1] = default(int  );
      next--;
    }
  }

  public int Count{
    get{
      return next;
    }
  }

  // Is this set a subset of other
  public bool Subset(IntSet other){
    foreach(int  e in this)
      if (!other.Member(e))
        return false;
    return true;
  }

  public override string ToString(){
    string elRes = "";
    for(int idx = 0; idx < next; idx++)
      elRes += " " + store[idx];
    return "{" + elRes + " "+ "}";
  }

  private bool Full{
    get{
      return next == capacity;
    }
  }

  public IEnumerator GetEnumerator (){
    return new SetEnumerator(this);
  }

  private class SetEnumerator: IEnumerator{

    private readonly IntSet set;
    private int idx;

    public SetEnumerator (IntSet s){
      this.set = s;
      idx = -1;   // position enumerator outside range
    }

    public Object Current{
      get {
       return set.store[idx];
      }
    }

    public bool MoveNext(){
      if (idx < set.next - 1){
        idx++;
        return true;
      }
      else
         return false;
    }

    public void Reset(){
      idx = -1;
    }

    public void Dispose(){
    }

  }

}
```

Program 41.1 *The class IntSet.*

The class IntSet is an example of an everyday implementation of integer sets. We have not attempted to come up with a clever representation that allows for fast set operations. The IntSet class is good enough for small sets. If you are going to work on sets with many elements, you should use a set class of better quality.

We chose to represent the elements in an integer array. We keep track of the position where to insert the next element (by use of the instance variable next). If there is not enough room in the array, we use the Array.Resize operation to make it larger. We delete elements from the set by shifting elements in the array 'to the left', in order to avoid wasted space. This approach is fairly expensive, but it is good enough for our purposes. The IntSet class is equipped with a GetEnumerator method, which returns an iterator. (We encountered iterators (enumerators) in the Interval class studied in Section 21.3. See also Section 31.6 for details on iterators. The GetEnumerator details are not shown in the paper version). The enumerator allows for traversal of all elements of the set with a **foreach** control structure.

A set is only, in a minimal sense, dependent on the types of elements (in our case, the type int). It does not even matter if the type of elements is a value type or a reference type (see Section 14.1 and Section 13.1 respectively). We do, however, apply equality on the elements, via use of the Equals method.

Nevertheless, the type int occurs many times in the class definition of IntSet. We have emphasized occurrences of int with color marks in Program 41.1.

```
1  using System;
2  using System.Collections;
3
4  class App{
5
6   public static void Main(){
7     IntSet s1 = new IntSet(),
8            s2 = new IntSet();
9
10    s1.Insert(1); s1.Insert(2);  s1.Insert(3);
11    s1.Insert(4); s1.Insert(5);  s1.Insert(6);
12    s1.Insert(5); s1.Insert(6);  s1.Insert(8);
13    s1.Delete(3); s1.Delete(6);  s1.Insert(9);
14
15    s2.Insert(8); s2.Insert(9);
16
17    Console.WriteLine("s1: {0}", s1);
18    Console.WriteLine("s2: {0}", s2);
19
20 // Exercises:
21 // Console.WriteLine("{0}", s2.Intersection(s1));
22 // Console.WriteLine("{0}", s2.Union(s1));
23 // Console.WriteLine("{0}", s2.Diff(s1));
24
25    if (s1.Subset(s2))
26      Console.WriteLine("s1 is a subset of s2");
27    else
28      Console.WriteLine("s1 is not a subset of s2");
29
30    if (s2.Subset(s1))
31      Console.WriteLine("s2 is a subset of s1");
32    else
33      Console.WriteLine("s2 is not a subset of s1");
34  }
35 }
```

Program 41.2 *A client of IntSet.*

In Program 41.2 we see a sample application of class IntSet. We establish two empty integer sets s1 and s2, we insert some numbers into these, and we try out some of the set operations on them. The comment lines 20-23 make use of set operations which will be implemented in Exercise 11.1. The output of Program 41.2 confirms that s2 is a subset of s1. The program output is shown in Listing 41.3 (only on web).

```
s1: { 1 2 4 5 8 9 }
s2: { 8 9 }
s1 is not a subset of s2
s2 is a subset of s1
```

Listing 41.3 *Output from the IntSet client program.*

We will now assume that we, a couple of days after we have programmed class IntSet, realize a need of class StringSet. Too bad! Class StringSet is almost like IntSet. But instead of occurrences of int we have occurrences of string.

We know how bad it is to copy the source text of IntSet to a new file called StringSet, and to globally replace 'int' with 'string'. When we need to modify the set class, all our modifications will have do be done twice!

For illustrative purposes - and despite the observation just described - we have made the class StringSet, see Program 41.4 (only on web). We have also replicated the client program, in Program 41.5 (only on web) and the program output in Listing 41.6 (only on web).

```
using System;
using System.Collections;

public class StringSet {

  private int capacity;
  private static int DefaultCapacity = 10;
  private string[] store;
  private int next;

  public StringSet(int capacity){
    this.capacity = capacity;
    store = new string[capacity];
    next = 0;
  }

  public StringSet(): this(DefaultCapacity){
  }

  public StringSet(string[] elements): this(elements.Length){
    foreach(string el in elements) this.Insert(el);
  }

  // Copy constructor
  public StringSet(StringSet s): this(s.capacity){
    foreach(string el in s) this.Insert(el);
  }

  public bool Member(string element){
    for(int idx = 0; idx < next; idx++)
      if (element.Equals(store[idx]))
        return true;
    return false;
  }

  public void Insert(string element){
    if (!this.Member(element)){
      if (this.Full){
        Console.WriteLine("[Resize to {0}]", capacity * 2);
        Array.Resize<string>(ref store, capacity * 2);
        capacity = capacity * 2;
      }
      store[next] = element;
      next++;
    }
  }

  public void Delete(string element){
    bool found = false;
    int  foundIdx = 0;
    for(int idx = 0; !found && (idx < next); idx++){
      if (element.Equals(store[idx])){
         found = true;
         foundIdx = idx;
      }
    }
    if (found){   // shift remaining elements left
      for(int idx = foundIdx+1; idx < next; idx++)
        store[idx-1] = store[idx];
      store[next-1] = default(string);
      next--;
    }
  }

  public int Count{
    get{
      return next;
    }
  }

  // Is this set a subset of other
  public bool Subset(StringSet other){
    foreach(string e in this)
      if (!other.Member(e))
         return false;
    return true;
  }

  private bool Full{
    get{
      return next == capacity;
    }
  }

  public IEnumerator GetEnumerator (){
    return new SetEnumerator(this);
  }

  private class SetEnumerator: IEnumerator{

    private readonly StringSet set;
    private int idx;

    public SetEnumerator (StringSet s){
      this.set = s;
      idx = -1;   // position enumerator outside range
    }

    public Object Current{
      get {
       return set.store[idx];
      }
    }

    public bool MoveNext(){
      if (idx < set.next - 1){
        idx++;
        return true;
      }
      else
         return false;
    }

    public void Reset(){
      idx = -1;
    }

    public void Dispose(){
    }

  }

  public override string ToString(){
    string elRes = "";
    for(int idx = 0; idx < next; idx++)
      elRes += " " + store[idx];
    return "{" + elRes + " "+ "}";
  }


}
```

Program 41.4   *The class StringSet.*

```
using System;
using System.Collections;

class App{

 public static void Main(){
   StringSet s1 = new StringSet(),
             s2 = new StringSet();

   s1.Insert("a"); s1.Insert("b");  s1.Insert("c");
   s1.Insert("dd"); s1.Insert("ee");  s1.Insert("ff");
   s1.Insert("ggg"); s1.Insert("hhh");  s1.Insert("iii");
   s1.Delete("c"); s1.Delete("ff");  s1.Insert("iii");

   s2.Insert("a"); s2.Insert("iii");

   Console.WriteLine("s1: {0}", s1);
   Console.WriteLine("s2: {0}", s2);

// Exercises:
// Console.WriteLine("{0}", s2.Intersection(s1));
// Console.WriteLine("{0}", s2.Union(s1));
// Console.WriteLine("{0}", s2.Diff(s1));

   if (s1.Subset(s2))
     Console.WriteLine("s1 is a subset of s2");
   else
     Console.WriteLine("s1 is not a subset of s2");

   if (s2.Subset(s1))
     Console.WriteLine("s2 is a subset of s1");
   else
     Console.WriteLine("s2 is not a subset of s1");
 }
}
```

Program 41.5 *A client of StringSet.*

```
s1: { a b dd ee ggg hhh iii }
s2: { a iii }
s1 is not a subset of s2
s2 is a subset of s1
```

Listing 41.6 *Output from the IntSet client program.*

41.3. The class ObjectSet

In Section 41.2 we learned the following lesson:

There is an endless number of *Type*Set classes. One for each *Type*. Each of them is similar to the others.

We will now review the solution to the problem which was used in Java before version 1.5, and in C# before version 2. These are the versions of the two languages prior to the introduction of generics.

The idea is simple: We implement a set class of element type Object. We call it ObjectSet. The type Object is the most general type in the type system (see Section 28.2). All other types inherit from the class Object.

Below, in Program 41.7 we show the class ObjectSet. In the paper version, only an outline with a few constructors and methods is included. The web version shows the full definition of class ObjectSet.

```
using System;
using System.Collections;

public class ObjectSet {

  private int capacity;
  private static int DefaultCapacity = 10;
  private Object[] store;
  private int next;

  public ObjectSet(int capacity){
    this.capacity = capacity;
    store = new Object[capacity];
    next = 0;
  }

  public ObjectSet(): this(DefaultCapacity){
  }

  public ObjectSet(Object[] elements): this(elements.Length){
    foreach(Object el in elements) this.Insert(el);
  }

  // Copy constructor
  public ObjectSet(ObjectSet s): this(s.capacity){
    foreach(Object  el in s) this.Insert(el);
  }

  public bool Member(Object  element){
    for(int idx = 0; idx < next; idx++)
      if (element.Equals(store[idx]))
        return true;
    return false;
  }

  public void Insert(Object  element){
    if (!this.Member(element)){
      if (this.Full){
        Console.WriteLine("[Resize to {0}]", capacity * 2);
        Array.Resize<Object>(ref store, capacity * 2);
        capacity = capacity * 2;
      }
      store[next] = element;
      next++;
    }
  }

  public void Delete(Object  element){
    bool found = false;
    int  foundIdx = 0;
    for(int idx = 0; !found && (idx < next); idx++){
      if (element.Equals(store[idx])){
         found = true;
         foundIdx = idx;
      }
    }
    if (found){   // shift remaining elements left
      for(int idx = foundIdx+1; idx < next; idx++)
        store[idx-1] = store[idx];
      store[next-1] = default(Object  );
      next--;
    }
  }

  public int Count{
    get{
      return next;
    }
  }

  // Is this set a subset of other
  public bool Subset(ObjectSet other){
    foreach(Object  e in this)
      if (!other.Member(e))
         return false;
    return true;
  }

  public override string ToString(){
    string elRes = "";
    for(int idx = 0; idx < next; idx++)
      elRes += " " + store[idx];
    return "{" + elRes + " "+ "}";
  }

  private bool Full{
    get{
      return next == capacity;
    }
  }

  public IEnumerator GetEnumerator (){
    return new SetEnumerator(this);
  }

  private class SetEnumerator: IEnumerator{

    private readonly ObjectSet set;
    private int idx;

    public SetEnumerator (ObjectSet s){
      this.set = s;
      idx = -1;   // position enumerator outside range
    }

    public Object  Current{
      get {
       return set.store[idx];
      }
    }

    public bool MoveNext(){
      if (idx < set.next - 1){
        idx++;
        return true;
      }
      else
         return false;
    }

    public void Reset(){
      idx = -1;
    }

    public void Dispose(){
    }

  }

}
```

Program 41.7 *The class ObjectSet.*

We can now write programs with a set of Die, a set of BankAccount, a set of int, etc. In Program 41.8 (only on web) we show a program, similar to Program 41.2, which illustrates sets of Die objects. (The class Die can be found in Section 10.1).

```
using System;
using System.Collections;

class App{

 public static void Main(){
   ObjectSet s1 = new ObjectSet(),
             s2 = new ObjectSet();

   Die d1 = new Die(6),  d2 = new Die(10),
       d3 = new Die(16), d4 = new Die(8);

   s1.Insert(d1); s1.Insert(d2);  s1.Insert(d3);
   s1.Insert(d4); s1.Insert(d1);  s1.Insert(d1);
   s1.Delete(d1); s1.Delete(d2);

   s2.Insert(d3); s2.Insert(d3); s2.Insert(d4);

   Console.WriteLine("s1: {0}", s1);
   Console.WriteLine("s2: {0}", s2);

// Exercises:
// Console.WriteLine("{0}", s2.Intersection(s1));
// Console.WriteLine("{0}", s2.Union(s1));
// Console.WriteLine("{0}", s2.Diff(s1));

   if (s1.Subset(s2))
     Console.WriteLine("s1 is a subset of s2");
   else
     Console.WriteLine("s1 is not a subset of s2");

   if (s2.Subset(s1))
     Console.WriteLine("s2 is a subset of s1");
   else
     Console.WriteLine("s2 is not a subset of s1");
 }
}
```

Program 41.8 *A client of ObjectSet - working with sets of Die objects.*

```
s1: { Die[16]:2 Die[8]:1 }
s2: { Die[16]:2 Die[8]:1 }
s1 is a subset of s2
s2 is a subset of s1
```

Listing 41.9 *Output from the ObjectSet client program.*

The main problem with class ObjectSet is illustrated below in Program 41.10. In line 12-20 we make a set of dice (s1), a set of integers (s2), a set of strings (s3), and set of mixed objects (s4). Let us focus on s1. If we take a die out of s1 with the purpose of using a Die operation on it, we need to typecase the element to a Die. This is shown in line 23. From the compiler's point of view, all elements in the set s1 are instances of class Object. With the cast (Die)o in line 23, we guarantee that each element in the set is a Die. (If an integer or a playing card should sneak into the set, an exception will be thrown). - The output of the program is shown in Listing 41.11 (only on web).

```
1  using System;
2  using System.Collections;
3  
4  class App{
5  
6   public static void Main(){
7     Die d1 = new Die(6),  d2 = new Die(10),
8         d3 = new Die(16), d4 = new Die(8);
9     int sum = 0;
10    string netString = "";
11 
12    ObjectSet
13      s1 = new ObjectSet(                  // A set of dice
14             new Die[]{d1, d2, d3, d4}),
15      s2 = new ObjectSet(                  // A set of ints
16             new Object[]{1, 2, 3, 4}),
17      s3 = new ObjectSet(                  // A set of strings
18             new string[]{"a", "b", "c", "d"}),
19      s4 = new ObjectSet(                  // A set of mixed things...
20             new Object[]{new Die(6), "a", 7});
21 
22    foreach(Object o in s1){
23      ((Die)o).Toss();
24      Console.WriteLine("{0}", (Die)o);
25    }
26 
27    Console.WriteLine("--------------");
28 
29    // Alternative - illustrating built-in cast of foreach
30    foreach(Die d in s1){
31      d.Toss();
32      Console.WriteLine("{0}", d);
33    }
34    Console.WriteLine("--------------");
35 
36    foreach(Object o in s2)
37      sum += ((int)o);
38    Console.WriteLine(
39      "Sum: {0}", sum);
40    Console.WriteLine("--------------");
41 
42    foreach(Object o in s3)
43      netString += ((string)o);
44    Console.WriteLine(
45      "NetString: {0}", netString);
46    Console.WriteLine("--------------");
```

```

  foreach(Object o in s4)
    Console.WriteLine("{0}", o);

  Console.WriteLine("--------------");
 }
}
```

Program 41.10 *A client of ObjectSet - working with set of different types.*

```
Die[6]:6
Die[10]:10
Die[16]:15
Die[8]:8
--------------
Die[6]:3
Die[10]:4
Die[16]:6
Die[8]:3
--------------
Sum: 10
--------------
NetString: abcd
--------------
Die[6]:1
a
7
--------------
```

Listing 41.11 *Output from the the client of ObjectSet - the version that works with sets of different types.*

41.4. Problems

The classes IntSet, StringSet and ObjectSet suffer from both programming and type problems:

- Problems with **IntSet** and **StringSet**
    - Tedious to write both versions: *Copy and paste* programming.
    - Error prone to maintain both versions
- Problems with **ObjectSet**
    - Elements of the set must be downcasted in case we need to use some of their specialized operations
    - We can create an inhomogeneous set
        - A set of "apples" and "bananas"

Generic types, to be introduced in the following chapter, offer a type safe alternative to ObjectSet, in which we are able to avoid type casting.

A Java virtual machine.

A VirtualMachine represents a Java virtual machine to which this Java virtual machine has attached. The Java virtual machine to which it is attached is sometimes called the *target virtual machine*, or *target VM*. An application (typically a tool such as a managemet console or profiler) uses a VirtualMachine to load an agent into the target VM. For example, a profiler tool written in the Java Language might attach to a running application and load its profiler agent to profile the running application.

A VirtualMachine is obtained by invoking the attach method with an identifier that identifies the target virtual machine. The identifier is implementation-dependent but is typically the process identifier (or pid) in environments where each Java virtual machine runs in its own operating system process. Alternatively, a VirtualMachine instance is obtained by invoking the attach method with a VirtualMachineDescriptor obtained from the list of virtual machine descriptors returned by the list method. Once a reference to a virtual machine is obtained, the loadAgent, loadAgentLibrary, and loadAgentPath methods are used to load agents into target virtual machine. The loadAgent method is used to load agents that are written in the Java Language and deployed in a JAR file. (See java.lang.instrument for a detailed description on how these agents are loaded and started). The loadAgentLibrary and loadAgentPath methods are used to load agents that are deployed either in a dynamic library or statically linked into the VM and make use of the JVM Tools Interface.

In addition to loading agents a VirtualMachine provides read access to the system properties in the target VM. This can be useful in some environments where properties such as java.home, os.name, or os.arch are used to construct the path to agent that will be loaded into the target VM.

The following example demonstrates how VirtualMachine may be used:

```
    // attach to target VM
VirtualMachine vm = VirtualMachine.attach("2177");

// start management agent
Properties props = new Properties();
props.put("com.sun.management.jmxremote.port", "5000");
vm.startManagementAgent(props);

// detach
vm.detach();
```

In this example we attach to a Java virtual machine that is identified by the process identifier `2177`. Then the JMX management agent is started in the target process using the supplied arguments. Finally, the client detaches from the target VM.

A VirtualMachine is safe for use by multiple concurrent threads.

Since:

1.6

Constructor Summary

All Methods Static Methods Instance Methods Abstract Methods Concrete Methods

Modifier and Type	Method and Description
static VirtualMachine	attach(String id) Attaches to a Java virtual machine.
static VirtualMachine	attach(VirtualMachineDescriptor vmd) Attaches to a Java virtual machine.
abstract void	detach() Detach from the virtual machine.
boolean	equals(Object ob) Tests this VirtualMachine for equality with another object.
abstract Properties	getAgentProperties() Returns the current *agent properties* in the target virtual machine.
abstract Properties	getSystemProperties() Returns the current system properties in the target virtual machine.
int	hashCode() Returns a hash-code value for this VirtualMachine.
String	id() Returns the identifier for this Java virtual machine.
static List<VirtualMachineDescriptor>	list() Return a list of Java virtual machines.
void	loadAgent(String agent) Loads an agent.
abstract void	loadAgent(String agent, String options) Loads an agent.
void	loadAgentLibrary(String agentLibrary) Loads an agent library.
abstract void	loadAgentLibrary(String agentLibrary, String options) Loads an agent library.
void	loadAgentPath(String agentPath) Load a native agent library by full pathname.
abstract void	loadAgentPath(String agentPath, String options) Load a native agent library by full pathname.
AttachProvider	provider()

	Returns the provider that created this virtual machine.
abstract String	**startLocalManagementAgent**() Starts the local JMX management agent in the target virtual machine.
abstract void	**startManagementAgent**(**Properties** agentProperties) Starts the JMX management agent in the target virtual machine.
String	**toString**() Returns the string representation of the VirtualMachine.

Constructors

Modifier	Constructor and Description
protected	VirtualMachine(AttachProvider provider, String id) Initializes a new instance of this class.

Method Summary

Methods inherited from class java.lang.Object

`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

Constructor Detail

VirtualMachine

-
  ```
  protected VirtualMachine(AttachProvider provider,
                           String id)
  ```

  Initializes a new instance of this class.

  Parameters:
  `provider` - The attach provider creating this class.

  `id` - The abstract identifier that identifies the Java virtual machine.

  Throws:
  `NullPointerException` - If provider or id is null.

- ## *Method Detail*

list

```
public static List<VirtualMachineDescriptor> list()
```

Return a list of Java virtual machines.

This method returns a list of Java VirtualMachineDescriptor elements. The list is an aggregation of the virtual machine descriptor lists obtained by invoking the listVirtualMachines method of all installed attach providers. If there are no Java virtual machines known to any provider then an empty list is returned.

Returns:

The list of virtual machine descriptors.

- **attach**

- `public static VirtualMachine attach(String id)`
- `throws AttachNotSupportedException, IOException`

Attaches to a Java virtual machine.

This method obtains the list of attach providers by invoking the AttachProvider.providers() method. It then iterates overs the list and invokes each provider's attachVirtualMachine method in turn. If a provider successfully attaches then the iteration terminates, and the VirtualMachine created by the provider that successfully attached is returned by this method. If the attachVirtualMachine method of all providers throws AttachNotSupportedException then this method also throws AttachNotSupportedException. This means that AttachNotSupportedException is thrown when the identifier provided to this method is invalid, or the identifier corresponds to a Java virtual machine that does not exist, or none of the providers can attach to it. This exception is also thrown if AttachProvider.providers() returns an empty list.

Parameters:

`id - The abstract identifier that identifies the Java virtual machine.`

Returns:

`A VirtualMachine representing the target VM.`

Throws:

`SecurityException - If a security manager has been installed and it denies AttachPermission ("attachVirtualMachine"), or another permission required by the implementation.`

`AttachNotSupportedException - If the attachVirtualmachine method of all installed providers throws AttachNotSupportedException, or there aren't any providers installed.`

`IOException - If an I/O error occurs`

`NullPointerException - If id is null.`

- **attach**

- `public static VirtualMachine attach(VirtualMachineDescriptor vmd)`
- `throws AttachNotSupportedException, IOException`

Attaches to a Java virtual machine.

This method first invokes the provider() method of the given virtual machine descriptor to obtain the attach provider. It then invokes the attach provider's attachVirtualMachine to attach to the target VM.

Parameters:
`vmd` - The virtual machine descriptor.

Returns:
A VirtualMachine representing the target VM.

Throws:
SecurityException - If a security manager has been installed and it denies AttachPermission ("attachVirtualMachine"), or another permission required by the implementation.

AttachNotSupportedException - If the attach provider's attachVirtualmachine throws AttachNotSupportedException.

IOException - If an I/O error occurs

NullPointerException - If vmd is null.

- **detach**

-
  ```
  public abstract void detach()
                       throws IOException
  ```

Detach from the virtual machine.

After detaching from the virtual machine, any further attempt to invoke operations on that virtual machine will cause an IOException to be thrown. If an operation (such as loadAgent for example) is in progress when this method is invoked then the behaviour is implementation dependent. In other words, it is implementation specific if the operation completes or throws IOException.

If already detached from the virtual machine then invoking this method has no effect.

Throws:
IOException - If an I/O error occurs

- **provider**

```
public final AttachProvider provider()
```

Returns the provider that created this virtual machine.

Returns:
The provider that created this virtual machine.

- **id**

```
public final String id()
```

Returns the identifier for this Java virtual machine.

Returns:
The identifier for this Java virtual machine.

- **loadAgentLibrary**

```
public
abstract void loadAgentLibrary(String agentLibrary,
                                String options)
                         throws AgentLoadException,
AgentInitializationException,
                                IOException
```

Loads an agent library.
A JVM TI client is called an *agent*. It is developed in a native language. A JVM TI agent is deployed in a platform specific manner but it is typically the platform equivalent of a dynamic library. Alternatively, it may be statically linked into the VM. This method causes the given agent library to be loaded into the target VM (if not already loaded or if not statically linked into the VM). It then causes the target VM to invoke the Agent_OnAttach function or, for a statically linked agent named 'L', the Agent_OnAttach_L function as specified in the JVM Tools Interface specification. Note that the Agent_OnAttach[_L] function is invoked even if the agent library was loaded prior to invoking this method.

The agent library provided is the name of the agent library. It is interpreted in the target virtual machine in an implementation-dependent manner. Typically an implementation will expand the library name into an operating system specific file name. For example, on UNIX systems, the name L might be expanded to libL.so, and located using the search path specified by the LD_LIBRARY_PATH environment variable. If the agent named 'L' is statically linked into the VM then the VM must export a function named Agent_OnAttach_L.

If the Agent_OnAttach[_L] function in the agent library returns an error then an AgentInitializationException is thrown. The return value from the Agent_OnAttach[_L] can then be obtained by invoking the returnValue method on the exception.

Parameters:

`agentLibrary` - The name of the agent library.

`options` - The options to provide to the Agent_OnAttach[_L] function (can be null).

Throws:

`AgentLoadException` - If the agent library does not exist, the agent library is not statically linked with the VM, or the agent library cannot be loaded for another reason.

`AgentInitializationException` - If the Agent_OnAttach[_L] function returns an error.

`IOException` - If an I/O error occurs

`NullPointerException` - If agentLibrary is null.

See Also:

`AgentInitializationException.returnValue()`

- **loadAgentLibrary**

```
public void loadAgentLibrary(String agentLibrary)
```

```
•                                  throws AgentLoadException,
•
   AgentInitializationException,
                                         IOException
```

Loads an agent library.

This convenience method works as if by invoking:

```
loadAgentLibrary(agentLibrary, null);
```

Parameters:
`agentLibrary` - The name of the agent library.

Throws:
`AgentLoadException` - If the agent library does not exist, the agent library is not statically linked with the VM, or the agent library cannot be loaded for another reason.

`AgentInitializationException` - If the Agent_OnAttach[_L] function returns an error.

`IOException` - If an I/O error occurs

`NullPointerException` - If agentLibrary is null.

- **loadAgentPath**

```
•  public abstract void loadAgentPath(String agentPath,
•                                     String options)
•                              throws AgentLoadException,
•
   AgentInitializationException,
                                      IOException
```

Load a native agent library by full pathname.
A JVM TI client is called an *agent*. It is developed in a native language. A JVM TI agent is deployed in a platform specific manner but it is typically the platform equivalent of a dynamic library. Alternatively, the native library specified by the agentPath parameter may be statically linked with the VM. The parsing of the agentPath parameter into a statically linked library name is done in a platform specific manner in the VM. For example, in UNIX, an agentPath parameter of /a/b/libL.so would name a library 'L'. See the JVM TI Specification for more details. This method causes the given agent library to be loaded into the target VM (if not already loaded or if not statically linked into the VM). It then causes the target VM to invoke the Agent_OnAttach function or, for a statically linked agent named 'L', the Agent_OnAttach_L function as specified in the JVM Tools Interface specification. Note that the Agent_OnAttach[_L] function is invoked even if the agent library was loaded prior to invoking this method.

The agent library provided is the absolute path from which to load the agent library. Unlike loadAgentLibrary, the library name is not expanded in the target virtual machine.

If the Agent_OnAttach[_L] function in the agent library returns an error then an AgentInitializationException is thrown. The return value from the Agent_OnAttach[_L] can then be obtained by invoking the returnValue method on the exception.

Parameters:

agentPath - The full path of the agent library.

options - The options to provide to the Agent_OnAttach[_L] function (can be null).

Throws:

AgentLoadException - If the agent library does not exist, the agent library is not statically linked with the VM, or the agent library cannot be loaded for another reason.

AgentInitializationException - If the Agent_OnAttach[_L] function returns an error.

IOException - If an I/O error occurs

NullPointerException - If agentPath is null.

See Also:

AgentInitializationException.returnValue()

- **loadAgentPath**

```
public void loadAgentPath(String agentPath)
                   throws AgentLoadException,
                          AgentInitializationException,
                          IOException
```

Load a native agent library by full pathname.

This convenience method works as if by invoking:

```
loadAgentPath(agentLibrary, null);
```

Parameters:

agentPath - The full path to the agent library.

Throws:

AgentLoadException - If the agent library does not exist, the agent library is not statically linked with the VM, or the agent library cannot be loaded for another reason.

AgentInitializationException - If the Agent_OnAttach[_L] function returns an error.

IOException - If an I/O error occurs

NullPointerException - If agentPath is null.

- **loadAgent**

```
public abstract void loadAgent(String agent,
                               String options)
                        throws AgentLoadException,
AgentInitializationException,
                               IOException
```

Loads an agent.

The agent provided to this method is a path name to a JAR file on the file system of the target virtual machine. This path is passed to the target virtual machine where it is interpreted. The target virtual machine attempts to start the agent as specified by the java.lang.instrument specification. That is, the specified JAR file is added to the system class path (of the target virtual machine), and the agentmain method of the agent class, specified by the Agent-Class attribute in the JAR manifest, is invoked. This method completes when the agentmain method completes.

Parameters:

`agent` - Path to the JAR file containing the agent.

`options` - The options to provide to the agent's agentmain method (can be null).

Throws:

`AgentLoadException` - If the agent does not exist, or cannot be started in the manner specified in the `java.lang.instrument` specification.

`AgentInitializationException` - If the agentmain throws an exception

`IOException` - If an I/O error occurs

`NullPointerException` - If agent is null.

- **loadAgent**

```
public void loadAgent(String agent)
               throws AgentLoadException,
                      AgentInitializationException,
                      IOException
```

Loads an agent.

This convenience method works as if by invoking:

```
loadAgent(agent, null);
```

Parameters:

`agent` - Path to the JAR file containing the agent.

Throws:

`AgentLoadException` - If the agent does not exist, or cannot be started in the manner specified in the `java.lang.instrument` specification.

`AgentInitializationException` - If the agentmain throws an exception

`IOException` - If an I/O error occurs

`NullPointerException` - If agent is null.

- **getSystemProperties**

```
public abstract Properties getSystemProperties()
```

throws IOException

Returns the current system properties in the target virtual machine.

This method returns the system properties in the target virtual machine. Properties whose key or value is not a String are omitted. The method is approximately equivalent to the invocation of the method System.getProperties in the target virtual machine except that properties with a key or value that is not a String are not included.

This method is typically used to decide which agent to load into the target virtual machine with loadAgent, or loadAgentLibrary. For example, the java.home or user.dir properties might be use to create the path to the agent library or JAR file.

Returns:

`The system properties`

Throws:

`AttachOperationFailedException - If the target virtual machine is unable to complete the attach operation. A more specific error message will be given by Throwable.getMessage().`

`IOException - If an I/O error occurs, a communication error for example, that cannot be identified as an error to indicate that the operation failed in the target VM.`

See Also:

`System.getProperties(), loadAgentLibrary(java.lang.String, java.lang.String), loadAgent(java.lang.String, java.lang.String)`

- **getAgentProperties**

-
  ```
  public abstract Properties getAgentProperties()
                                   throws IOException
  ```

Returns the current *agent properties* in the target virtual machine.

The target virtual machine can maintain a list of properties on behalf of agents. The manner in which this is done, the names of the properties, and the types of values that are allowed, is implementation specific. Agent properties are typically used to store communication end-points and other agent configuration details. For example, a debugger agent might create an agent property for its transport address.

This method returns the agent properties whose key and value is a String. Properties whose key or value is not a String are omitted. If there are no agent properties maintained in the target virtual machine then an empty property list is returned.

Returns:

`The agent properties`

Throws:

`AttachOperationFailedException - If the target virtual machine is unable to complete the attach operation. A more specific error message will be given by Throwable.getMessage().`

`IOException - If an I/O error occurs, a communication error for example, that cannot be identified as an error to indicate that the operation failed in the target VM.`

- **startManagementAgent**

-
  ```
  public
  abstract void startManagementAgent(Properties agentProperties)
                                throws IOException
  ```

Starts the JMX management agent in the target virtual machine.

The configuration properties are the same as those specified on the command line when starting the JMX management agent. In the same way as on the command line, you need to specify at least the com.sun.management.jmxremote.port property.

See the online documentation for Monitoring and Management Using JMX Technology for further details.

Parameters:

agentProperties - A Properties object containing the configuration properties for the agent.

Throws:

AttachOperationFailedException - If the target virtual machine is unable to complete the attach operation. A more specific error message will be given by Throwable.getMessage().

IOException - If an I/O error occurs, a communication error for example, that cannot be identified as an error to indicate that the operation failed in the target VM.

IllegalArgumentException - If keys or values in agentProperties are invalid.

NullPointerException - If agentProperties is null.

Since:

1.8

- **startLocalManagementAgent**

-
  ```
  public abstract String startLocalManagementAgent()
                                     throws
  IOException
  ```

Starts the local JMX management agent in the target virtual machine.

See the online documentation for Monitoring and Management Using JMX Technology for further details.

Returns:

The String representation of the local connector's service address. The value can be parsed by the JMXServiceURL constructor.

Throws:

AttachOperationFailedException - If the target virtual machine is unable to complete the attach operation. A more specific error message will be given by Throwable.getMessage().

IOException - If an I/O error occurs, a communication error for example, that cannot be identified as an error to indicate that the operation failed in the target VM.

Since:

`1.8`

- **hashCode**

```
public int hashCode()
```

Returns a hash-code value for this VirtualMachine. The hash code is based upon the VirtualMachine's components, and satifies the general contract of the Object.hashCode method.

Overrides:

`hashCode in class Object`

Returns:

`A hash-code value for this virtual machine`

See Also:

`Object.equals(java.lang.Object), System.identityHashCode(java.lang.Object)`

- **equals**

```
public boolean equals(Object ob)
```

Tests this VirtualMachine for equality with another object.

If the given object is not a VirtualMachine then this method returns false. For two VirtualMachines to be considered equal requires that they both reference the same provider, and their identifiers are equal.

This method satisfies the general contract of the Object.equals method.

Overrides:

`equals in class Object`

Parameters:

`ob - The object to which this object is to be compared`

Returns:

`true if, and only if, the given object is a VirtualMachine that is equal to this VirtualMachine.`

See Also:

`Object.hashCode(), HashMap`

- **toString**

```
public String toString()
```

Returns the string representation of the `VirtualMachine`.

Overrides:

`toString in class Object`

Returns:

`a string representation of the object.`

Reflection in Java is a special feature in the Java programming language that provides us with a way to get information regarding the class to which an object belongs and the methods of that class that can be executed using that object.

We can use Java Reflection to invoke these methods at runtime without even knowing its name and even change their behaviour at runtime.

To reflect a class in java, we need to create an object of **Class**. After that, we can use this class to gather information regarding the methods, fields, and constructors that are present in a class.

There are 3 main ways we can use to create an object of Class.

1. Using forName() methods

the forName() method takes in the class's name that is to be reflected as its argument.

For Example:

```
class Car {...}



// creating an object of the Class

// so that it can  reflect the car class

Class carClass = Class.forName("Car");
```

2. Using getClass() method

in this method, we use an object of our car class to make an object of the **Class**

Example:

```
// creating an object of the Car class

Car c1 = new Car();



// creating an object of the Class

// so that it can  reflect the car class

Class b = c1.getClass();
```

3. Using .class extension

In this method, we simply use .class after the name of our class.

Example:

```
// creating an object of the Class

// so that it can  reflect the car class

Class carClass = Car.class;
```

Now that we have learned how to make objects belonging to the **"Class"** This object can be used to obtain runtime details about the relevant class.

We are going to create a class named Vehicle and another subclass named Car and then we are going to inspect the Car class with the help of an object of Class.

```
import java.lang.Class;

import java.lang.reflect.*;



class Vehicle {

}



// We have to put this particular class in a unique Car.java file

public class Car extends Vehicle {

  public void display() {

    System.out.println("Car goes fast");

  }

}



//  you have to place this in the Main.java file

class Main {
```

```
public static void main(String[] args) {
 try {
   // creating an object in the class Car
   Car c1 = new Car();

   // creating an object of the type Class
   // using the getClass() method
   Class object = c1.getClass();

   // getting the name of our class
   String n = object.getName();
   System.out.println("Name is => " + n);

   // getting the access modifiers of our class
   int modi = object.getModifiers();

   // converting the access modifiers to string format
   String m = Modifier.toString(modi);
   System.out.println("Modifier is => " + m);

   // getting the superclass of our Car class
   Class sClass = object.getSuperclass();
   System.out.println("Superclass is => " + sClass.getName());
 }
```

```
    catch (Exception e) {

      e.printStackTrace();

    }

  }

}
```

Output:

```
Name is => Car

Modifier is =>  public

Superclass is => Vehicle
```

Let's see some of the methods of **Class** that we have used in this code

- object.getName() - This returns the name of the class that we are inspecting
- object.getModifiers() - This returns the access modifier of the class that we are inspecting
- object.getSuperclass() - This returns the superclass of the class that we are inspecting

If we want to change or modify the class members, the java.lang.reflect package provides some classes to do that. some of them are

- **Method class**
- **Field class**
- **Constructor class**

Let's move ahead and learn about these features of Java reflection.

What is Multithreading?

Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel. If you are working on a computer, it runs multiple applications and allocates processing power to them. A simple program runs in sequence and the code statements execute one by one. This is a single-threaded application. But, if the programming language supports creating multiple threads and passes them to the operating system to run in parallel, it’s called multithreading.

Multithreading vs Multiprocessing

When we talk about multithreading, we don’t care if the machine has a 2-core processor or a 16-core processor. Our work is to create a multithreaded application and let the OS handle the allocation and execution part. In short, multithreading has nothing to do with multiprocessing.

How does Java Support Multithreading?

Java has great support for multithreaded applications. Java supports multithreading through **Thread** class. Java Thread allows us to create a lightweight process that executes some tasks. We can create multiple threads in our program and start them. Java runtime will take care of creating machine-level instructions and work with OS to execute them in parallel.

What are the different types of threads?

There are two types of threads in an application - **user thread** and **daemon thread**. When we start an application, the **main** is the first user thread created. We can create multiple user threads as well as daemon threads. When all the user threads are executed, JVM terminates the program.

What is Thread Priority?

When we create a thread, we can assign its priority. We can set different priorities to different Threads but it doesn't guarantee that a higher priority thread will execute first than a lower priority thread. The thread scheduler is the part of Operating System implementation and when a Thread is started, its execution is controlled by Thread Scheduler and JVM doesn't have any control over its execution.

How Do we Create Thread in Java?

We can create Threads by either implementing **Runnable** interface or by extending **Thread** Class.

```
Thread t = new Thread(new Runnable(){

   @Override

   public void run() {

   }

});
```

Copy

Above is a one-line statement to create a new Thread. Here we are creating a Runnable as an anonymous class. If you are familiar with lambda expressions, we can create a Thread with much shorter code.

```
Runnable runnable = () -> System.out.println("Hello");
```

Copy

Once we have created a Thread, we have to start its execution by calling the start() method.

```
runnable.start();
```

Copy

I have written a lot of posts explaining the concepts of multithreading in Java. You can go through these in sequence to learn everything about multithreading, its real-life usage, thread lifecycle, thread pool, etc.

1. Java Thread and Runnable

This is the first post about the Thread class and Runnable interface. You will also learn about Process and Thread. What is the difference between Thread and Process? Benefits of using Threads and how we can create Threads using Runnable interface and Thread class. This post also compares the Runnable interface with the Thread class.

2. Java Thread Sleep

Java Thread sleep is used to pause the execution of the current thread. We will use Thread sleep extensively in future posts, so it's good to know how it works and is it accurate or not?

3. Java Thread Join

Sometimes we need to wait for other threads to finish their execution before we can proceed. We can achieve this using the **Thread join** method, learn how it works and when we should use it.

4. Java Thread States

Understanding different **states of thread** are important. Learn how a thread changes its state and how the operating system thread scheduler changes the state of a thread.

5. Java Thread wait, notify and notifyAll

Java Object class contains three methods to communicate the lock status of a resource. Learn with example usage of these Object class methods in a simple Wait-Notify implementation.

6. Thread Safety and Synchronization

We know that Threads share Object resources, which can lead to data corruption because these operations are not atomic. Learn how we can achieve thread-safety in java using different methods. Read this post to learn about the correct usage of synchronization, synchronized methods, and synchronized blocks.

7. Java Exception in thread main

JVM creates the first thread using the main method. This post explains some common exceptions we see in daily life and what is the root cause of them and how to fix them.

8. Thread Safety in Singleton Class

In this article, you will learn basic concepts of creating a Singleton class. What are thread safety issues with different implementations? How we can achieve thread-safety in Singleton class.

9. Daemon Thread in Java

A simple article explaining **daemon threads** and how we can create daemon threads in java.

10. Java Thread Local

We know that threads share Object's variables but what if we want to have thread-local variables created at the class level. Java provides the ThreadLocal utility class to create thread-local variables. Read more to learn about how we can create ThreadLocal variables in the java program.

11. Java Thread Dump

Java Thread dump provides the information of the current thread. A thread dump is useful to analyze performance issues with the application. You can use thread dump to find and fix deadlock situations. This post explains different methods that can be used to generate thread dumps in java.

12. How to Analyze Deadlock in Java

Deadlock is a situation where multiple threads are waiting for each other to release resources causing cyclic dependency. This article discusses the situation in which we can get deadlock in a java program. How we can use Thread dump to find the deadlock and best practices to avoid deadlock in java program.

13. Java Timer Thread

This post explains how we can use **Java Timer** and TimerTask classes to create jobs to run at a scheduled interval, an example program showing its usage, and how we can cancel the timer.

14. Java Producer Consumer Problem

Before Java 5, the producer-consumer problem can be solved using wait() and notify() methods but the introduction of BlockingQueue has made it very easy. Learn how we can use BlockingQueue to solve the producer-consumer problem in java.

15. Java Thread Pool

Java Thread Pool is a collection of worker threads waiting to process jobs. Java 5 introduction of the Executor framework has made it very easy to create a thread pool in java. We can use Executors and ThreadPoolExecutor classes to create and manage a thread pool.

16. Java Callable Future

Sometimes we want our Thread to return some values that we can use. Java 5 Callable can be used in that case, which is similar to the Runnable interface. We can use the Executor framework to execute Callable tasks.

17. Java FutureTask Example

FutureTask class is the base concrete class that implements the Future interface. We use it with Callable implementation and Executors for asynchronous processing. The FutureTask class provides implementation methods to check the state of the task and return the value to the calling program once its processing is finished. It comes in handy when you want to override some of the implementation methods of the Future interface.

What is JDBC?

Java™ database connectivity (JDBC) is the JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results.

Since JDBC is a standard specification, one Java program that uses the JDBC API can connect to any database management system (DBMS), as long as a driver exists for that particular DBMS.

Parent topic: Follow given link for detail information about JDBC.

Getting started :

https://www.ibm.com/docs/en/informix-servers/12.10?topic=ijdg-getting-started