Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

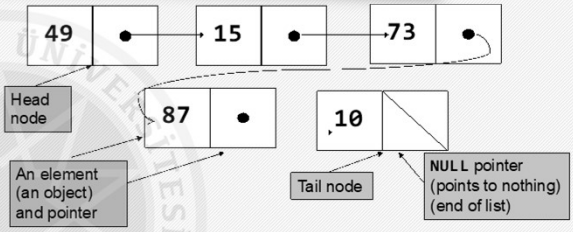
BLM2012 Nesneye Yönelik Programlama Hafta 9

Hafta	Tarih	Konular
1	01.03.2022	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2022	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2022	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2022	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2022	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
6	05.04.2022	NYP'da Özel Konular: Abstract Classes, Interfaces, Enum Sınıfları
7	12.04.2022	Exception Handling, Unit Test
8	21.04.2022	1. Ara Sınav (10:00-12:00)
9	26.04.2022	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları).
10	03.05.2022	Ramazan Bayramı
11	10.05.2022	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
12	17.05.2022	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
13	24.05.2022	2. Ara Sınav
14	31.05.2022	Paralel Programlamaya Giriş

Öğr. Grv. Furkan ÇAKMAK

GENERIC CLASSES and DATA STRUCTURES in JAVA

- Generic programming
- LinkedList
  - Self-referential class objects
  - The first (head) and the last (tail) nodes
  - Connected by pointer links
  - Iterator object
- Advantages of linked lists over arrays:
  - Enlarging a list costs nothing!
  - Insertion and removal of elements to any position is faster.
  - Sorting algorithms work faster on linked lists.
- Advantage of arrays over linked lists:
  - Lists are traversed sequentially where any ith member of an array is directly accessible.



GENERIC CLASSES and DATA STRUCTURES in JAVA

BLM2012 Nesneye Yönelik Programlama Hafta 9

- Types of linked lists:
  - Single-linked list: Only traversed in one direction
  - Doubly-linked list: Allows traversals both forwards and backwards
- A list may also be circular.
  - Pointer in the last node points back to the first node (like prayer beads)
- java.util.ArrayList (single-linked)
  - ArrayList myList = new ArrayList();

Öğr. Grv. Furkan ÇAKMAK

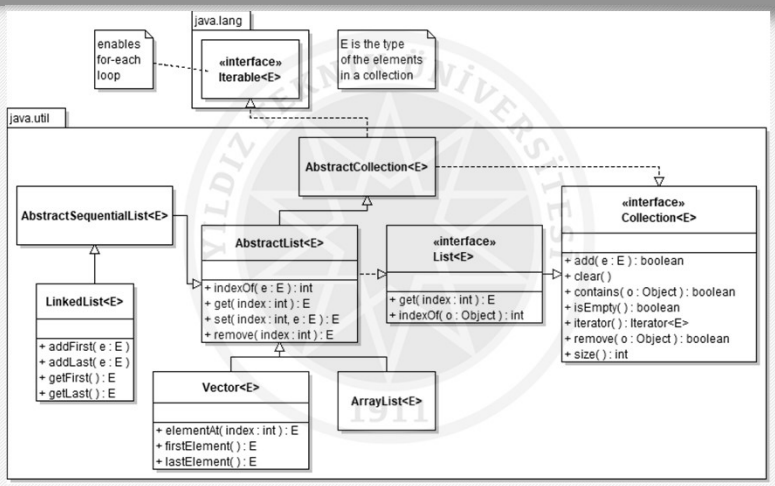
GENERIC CLASSES and DATA STRUCTURES in JAVA

BLM2012 Nesneye Yönelik Programlama Hafta 9

- Fundamental methods of the ArrayList class:
  - **add**(<T> object): Adds an element (an object of type T) to the end of the list.
  - <T> **get**(int i): Returns the ith element.
  - int **size**( ): Returns the number of elements in this list
- A selection of the other methods of the ArrayList class:
  - **ensureCapacity**(int size): Increases the capacity of this ArrayList instance, if necessary.
  - **trimToSize**( ): Trims the capacity of this ArrayList instance to be the list's current size.
  - **set**(int i, <T> element): Replaces the element at the specified position in this list with the specified element.
  - **remove**(int i): Removes the ith element from this list
  - *If the current size is less than i, an IndexOutOfBoundsException is thrown (unchecked).*

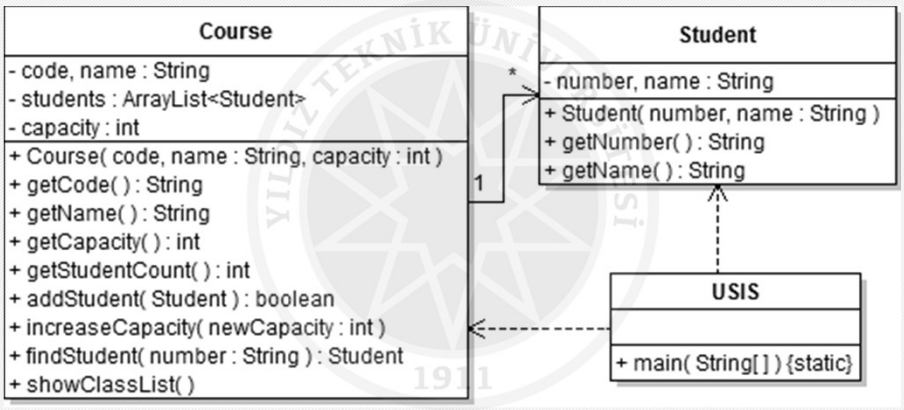
GENERIC CLASSES and DATA STRUCTURES in JAVA

BLM2012 Nesneye Yönelik Programlama Hafta 9



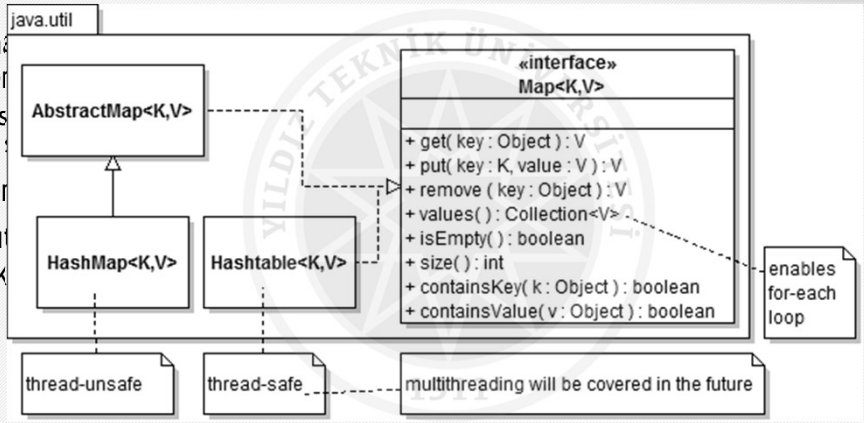
Öğr. Grv. Furkan ÇAKMAK

GENERIC CLASSES and DATA STRUCTURES in JAVA



MAP STRUCTURE in JAVA

- The ma
  elemer
  - This
    list
- Elemer
- java.ut
  - K: K



MAP STRUCTURE EXAMPLE in JAVA


```
package nyp10b;
import java.util.*;

public class Course {
    private String code; private String name; private int capacity;
    private HashMap<String,Student> students;

    public Course(String code, String name, int capacity) {
        this.code = code; this.name = name; this.capacity = capacity;
        students = new HashMap<String,Student>();
    }
    public String getCode() { return code; }
    public String getName() { return name; }
    public int getCapacity() { return capacity; }
    public int getStudentCount() {
        return students.size();
    }
    public boolean addStudent( Student aStudent ) {
        if( getStudentCount() == capacity ||
                findStudent(aStudent.getNumber()) != null )
            return false;
        students.put(aStudent.getNumber(), aStudent);
        return true;
    }
```


MAP STRUCTURE EXAMPLE in JAVA (CON'T)


```
    public Student findStudent( String number ) {
        return students.get(number);
    }
    public void increaseCapacity( int newCapacity ) {
        if( newCapacity <= capacity )
            return;
        capacity = newCapacity;
    }
    public void showClassList( ) {
        System.out.println("Class List of "+code+" "+name);
        System.out.println("Student#  Name, Surname");
        System.out.println("--------  ----------------------------");
        for( Student aStudent : students.values() )
            System.out.println(aStudent.getNumber()+
            " " + aStudent.getName());
    }
}
```

SUMMARY OF FUNDAMENTAL DATA STRUCTURE IMPLEMENTATIONS

- java.util.LinkedList<E> implements List<E>
  - Faster insertions and deletions
  - Slower random access
  - Doubly-linked (Can be traversed backwards by obtaining a ListIterator instance [not to be covered?]).
- java.util.ArrayList<E> implements List<E>
  - Slower insertions and deletions
  - Faster random access
- java.util.Vector<E> implements List<E>
  - Similar to ArrayList
  - synchronized = thread-safe
    - Suitable for multi-threaded use, slower in single-threaded use
- java.util.HashMap<K,V> implements Map<K,V>
  - Used for fast searches by a key (indexed)
- java.util.Hashtable<K,V> implements Map<K,V>
  - Similar to HashMap but synchronized
    - Suitable for multi-threaded use, slower in single-threaded use
  - Attention: Lowercase t in class name Hashtable

Sabırla Dinlediğiniz İçin Teşekkürler

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

BLM2012 Nesneye Yönelik Programlama Hafta 11

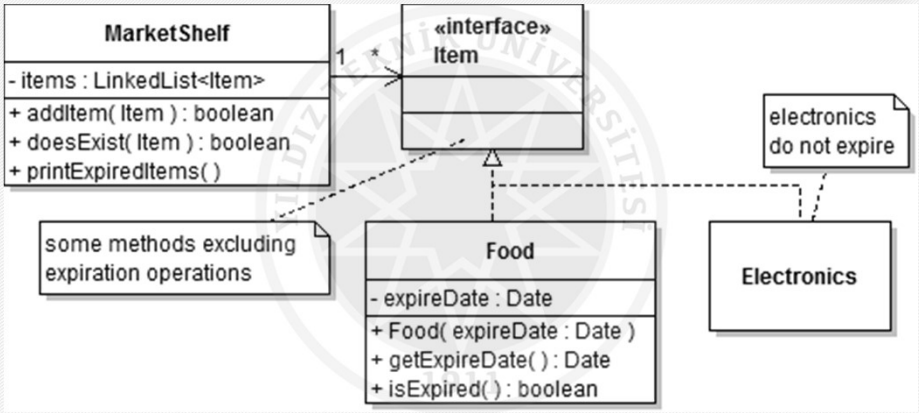
Hafta	Tarih	Konular
1	01.03.2022	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2022	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2022	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2022	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2022	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
6	05.04.2022	NYP'da Özel Konular: Abstract Classes, Interfaces, Enum Sınıfları
7	12.04.2022	Exception Handling, Unit Test
8	21.04.2022	1. Ara Sınav (10:00-12:00)
9	26.04.2022	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları).
10	03.05.2022	Ramazan Bayramı
11	10.05.2022	Tip dönüşümü, Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi), İç Sınıflar
12	17.05.2022	Paralel Programlamaya Giriş
13	24.05.2022	2. Ara Sınav
14	31.05.2022	GUI (Graphical User Interface) Kavramlarına Giriş

Öğr. Grv. Furkan ÇAKMAK

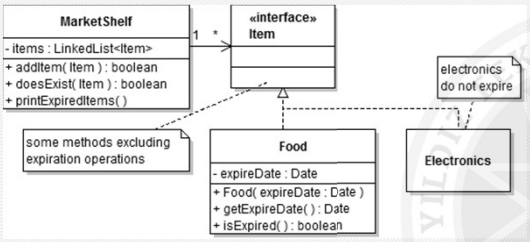
TYPECASTING

- An instance of a sub class can be used wherever an instance of its super class is expected.
  - Type-Safe Operation
- We can convert a specific object to a more general one without loosing any information.
- You can make a manual cast from one type to another, according to the following rules:
  - From the interface to the class of the object
  - From the super class to the sub class

TYPECASTING - EXAMPLE

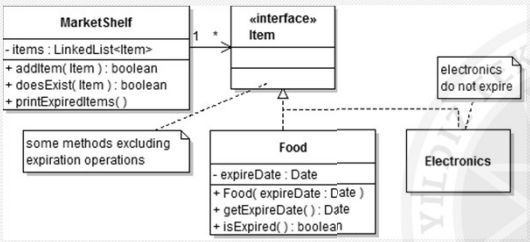


TYPECASTING – EXAMPLE (CON'T)



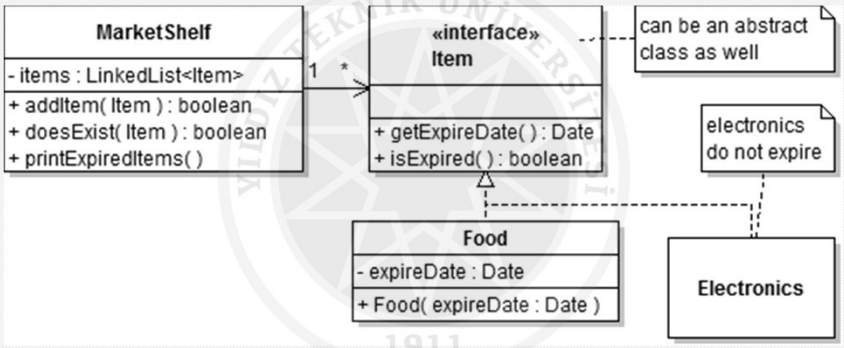

```
public void printExpiredItems( ) {
    boolean hasExpiredItem = false;
    System.out.print("Expired item(s): ");
    for( Item item : items )
      if( item instanceof Food )
        if( ((Food)item).isExpired() ) {
          System.out.print(item);
          hasExpiredItem = true;
        }
    if( hasExpiredItem == false )
      System.out.println("All items are fresh!");
  }
  public boolean addItem( Item anItem ) {
    if( doesExist(anItem) )
      return false;
    items.add(anItem);
    return true;
  }
```


TYPECASTING – EXAMPLE (CON'T)




```
public void printExpiredItems( ) {
    boolean hasExpiredItem = false;
    System.out.print("Expired item(s): ");
    for( Item item : items )
      if( item instanceof Food )
        if( ((Food)item).isExpired() ) {
          System.out.print(item);
          hasExpiredItem = true;
        }
    if( hasExpiredItem == false )
      System.out.println("All items are fresh!");
  }
  public boolean addItem( Item anItem ) {
    if( doesExist(anItem) )
      return false;
    items.add(anItem);
    return true;
  }
```

TYPECASTING – EXAMPLE (CON'T)



WORKING WITH FILES

- RELATED EXCEPTIONS
  - java.io.IOException: Represents I/O exceptions in general.
  - java.io.EOFException extends IOException: Indicates that the end of file or stream has been reached unexpectedly.
  - java.io.FileNotFoundException extends IOException: Indicates that the requested file cannot be found in the given path.
  - java.lang.SecurityException extends java.lang.RuntimeException: Indicates that the requested operation cannot be executed due to security constraints.
- File operations are separated into two main groups in Java:
  - File management: Operations such as creating, renaming, deleting files and folders.
  - I/O operations.

WORKING WITH FILES - FILE MANAGEMENT

- java.io.File
  - Files
  - Folders
- File(String fileName)
  - fileName should contain both the path and the name of the file/folder.
  - Full path vs. relative path.
- Path separator:
  - Windows uses \ (should be denoted as \\ in Strings), Unix uses /.
  - public static String File.separator
  - public static char File.separatorChar
- File(String path, String name) and File(File path, String name) constructors:

WORKING WITH FILES - FILE MANAGEMENT

BLM2012 Nesneye Yönelik Programlama Hafta 11

- Some methods of the class java.io.File:
  - boolean exists( ); tells whether the file exists or not.
  - boolean isFile( ); returns true if this File object represents a file, false otherwise, i.e. this object represents a folder.
  - File getParentFile( ); Returns the directory where this file/folder resides.
  - String getCanonicalPath( ) throws IOException; Returns the full path of the file/folder, including the file name.
  - boolean canRead( ); Can this application read form this file?
  - boolean canWrite( ); Can this application write to this file?
  - boolean createNewFile( ); Actually creates the file. Only for files!
  - boolean mkdir( ); Actually creates the folder. Only for folders!
  - boolean mkdirs( ); Actually creates the folder with all necessary parent folders. Only for folders!
  - boolean renameTo(File newName); Renames the file.
  - boolean delete( ); Deletes the file.

Öğr. Grv. Furkan ÇAKMAK

WORKING WITH FILES - I/O OPERATIONS

BLM2012
Nesneye
Yönelik
Programlama
Hafta 11

- I/O OPERATIONS USING STREAMS
  - Any I/O source is represented as stream in Java
    - Files, memory, command prompt, network, etc.
  - Binary vs. Text format:
    - Binary I/O is fast and efficient, but it is not easily readable by humans.
    - Text I/O is the opposite.
  - Random vs. Sequential access:
    - Sequential access: All records are accessed from the beginning to the end
    - Random access: A particular record can be accessed directly.
    - Disk files are random access, but streams of data from a network are not.

WORKING WITH FILES - I/O OPERATIONS

BLM2012
Nesneye
Yönelik
Programlama
Hafta 11

- Serialization – Output operations:
  - We will write entire objects to a file on disk.
  - java.io.Serializable
  - ObjectOutputStream and FileOutputStream objects are chained together for serialization.
  - About the transient keyword

WORKING WITH FILES - I/O OPERATIONS

- About the lines beginning with @ :
  - These are special commands called “annotations”.
  - They work at the “meta” level, i.e. they contain “information about information”.
  - They give information to the IDE, compiler, another programme, etc. about this program.
  - We have used the annotation mechanism to remove the warnings.
  - In fact, warnings must be taken into consideration.
  - Example: @SuppressWarnings("serial")

WORKING WITH FILES - I/O OPERATIONS - EXAMPLE

- Kod Gösterimi Yapılsın!

INNER CLASSES

- You can code a class within a class.
  - An inner class is coded within an outer class.
- An inner class can:
  - Access all members of the outer class, including the private ones.
  - Be hidden from other classes of the same package, if defined as private.
  - It is frequently used in form of anonymous inner classes in multithreaded and GUI programming.
    - Anonymous = without a name!
- You cannot:
  - define a static method in a an inner class.

INNER CLASSES - EXAMPLE

- Kod Gösterimi Yapılsın!

Sabırla Dinlediğiniz İçin Teşekkürler

BLM2012
Nesneye
Yönelik
Programlama
Hafta 11

Öğr. Grv. Furkan Çakmak