Java Arrays & ArrayList

Rui S. Moreira

Array is a class (array index starts at 0)

- **Array of primitive types/values**
  - **Declaration:**

```
//Do not creates array (just the reference/name of array)
char[] arrayChars; // ⇔ char arrayChars[];
int arrayInts[]; // ⇔ int[] arrayInts;
```

  - **Creation:**

```
// Allocates memory for holding the primitve values
arrayChars = new char[127]; // Values are set to '\u0000'
arrayPontos = new Ponto[10]; // Values are set to 0

// We may declare and create at the same time
char[] arrayChars = new char[127];
int arrayInts[] = new int[10];
```

  - **Initialisation:**

```
// Setting array values to 'a'
for (int i=0; i<arrayChars.length; i++){
    arrayChars[i]='a';
}
```

Array is a class (has attribute *length*)

- **Array of references to objects**
  - **Declaration:**

```
//Do not creates array (just the reference/name of array)
Ponto arrayPontos[]; // ⇔ Ponto[] arrayPontos;
```

  - **Creation:**

```
// Allocates memory for references to objects (not the objects)
arrayPontos = new Ponto[10]; Values are set to null
```

  - **Initialisation** (NB: we may not use/reference elements of array if they are not initialised first):

```
// Setting array references to real objects
for (int i=0; i<arrayPontos.length; i++){
    arrayPontos[i] = new Ponto(i, i);
}
```


Declaring, Creating and Initializing Arrays

```
// Create and initializes array of 4 strings
String arrayStr[] = {
            "John", "Mary", "Peter", "Paul"
      };

// Equivalent to
String arrayStr[] = new String[4];
arrayStr[0]=new String("John");
arrayStr[1]=new String("Mary");
arrayStr[2]=new String("Peter");
arrayStr[3]=new String("Paul");
```

Arrays

- **After creating an array we may not re-dimension it!**

- **We can, however, change the (array) reference variable to a new array:**

```
// Declaring, Creating and Initialising an array of 4 Strings
String arrayStr[] = {
            "John", "Mary", "Peter", "Paul"
      };

// Change the array reference to a new/bigger array
arrayStr = new String[10]; // We loose the previous values
```


Exercises

- Create Class `TestArrayApp`:
  - In main method declare and create 2 int arrays, with length 10 and 20;
  - Init array1 with their index values, e.g., *array1[i]=i;*
  - For each element of array1 calculate respective factorial and store it in array2, e.g., *array2[i]=MyMath.fact_for(array1[i]);*
  - Print out both arrays using different cycles;

  - In main method declare and create 2 arrays of *Ponto* and *Rectangle*, with length 20 and 10;
  - Initialise array1 with new point objects (each point with coordinates of the array index, e.g., ***arrayPoints[i] = new Ponto(i,i);***
  - Initialise array2 with new rectangle objects (each rectangle uses adjacent points of array1, e.g., ***arrayRect[i] = new Rectangle(arrayPoints[i], arrayPoints[i+1]);***
  - Print out both arrays using the *toString()* method available in the Ponto and Rectangle classes;

Arrays – System class

- ***System*** class provides usefull methods, **e.g., *arraycopy()***

```
int arrayOriginal[] = { 1, 2, 3 };
int arrayCopy[] = { 4, 5, 6, 7 };
int size = arrayOriginal.length;

// Copy arrayOriginal over arrayCopy (overwrite)
// Start reading at position 0 of arrayOriginal
// Start writing over position 2 of arrayCopy
// Stop copy when reach end/size arrayOriginal
System.arraycopy(arrayOriginal, 0, arrayCopy, 2, size);

// This for-cycle prints content of arrayCopy
// which now is { 4, 1, 2, 3 }
for (int i=0; i<arrayCopy.length; i++){
   System.out.print(" "+ arrayCopy[i]);
}
```


Multidimensional Arrays (Matrix)

- **Java do not provides multidimensional arrays**
- **but allows the declaration of arrays of arrays of arrays... ☺**

```
// Declaring and creating a rectangular matrix
int rectangularArray[][] = new int[4][5];

// Now we can set each element of the matrix ([line][column])
rectangularArray[0][2] = 18; // Sets line 0, column 2 element
rectangularArray[1][4] = 6; // Sets line 1, column 4 element
rectangularArray[4][5] = 6; // Runtime Exception (ArrayIndexOutOfBound)
```

Multidimensional Arrays (Matrix)

- **Java allows the declaration of non-rectangular arrays**

```
// Declares and creates array (lines) with 4 references
// to other arrays of ints, though these do not exist yet)
int twoDimArray[][] = new int[4][];

// Then we may create each array individually (column)
// each array may have different lengths
twoDimArray[0] = new int[1];
twoDimArray[1] = new int[2];
twoDimArray[2] = new int[3];
twoDimArray[3] = new int[4];

// NB: this is NOT LEGAL (compile-time error)
int twoDimArray[][] = new int[][4];
```


Exercises

- Create `MatrixInts` Class for storing matrixes of ints and with methods for:
  - `MatrixInts matrixAdd(MatrixInts m);` //Matrix addition
  - `MatrixInts matrixSub(MatrixInts m);` //Matrix subtraction
  - `MatrixInts matrixMult(MatrixInts m);` //Matrix multiplication
  - `MatrixInts matrixInv();` //Matrix inversion
  - ...
- Create `Scrambler` Class
  - Receives a *magic-word* (string) in the constructor;
  - Stores the characters in an *array of chars*;
  - Re-orders (*scrambles*) the characters to a new array and prints it;
  - Gives another user `n` shots (passed also in constructor) to guess the magic-word.

Vector and ArrayList classes

- Java provides utility classes for storing objects
  - `Vector` behaves like a growable arrays of `Object`
    - **`java.util.Vector<generic-type>`** class – **Deprecated**
  - `ArrayList` behaves like a list of `Object`
    - **`java.util.ArrayList<generic-type>`** class
  - Both can store any number of objects (NO primitive types allowed!)

```
// Declaring and creating collections of objects
Vector v1 = new Vector(), v2 = new Vector(20); //Initial size
v1.addElement("Hello"); //Store String
v1.addElement(new Integer(4)); //Store Integer

ArrayList<String> a1 = new ArrayList<>();//Just Strings
a1.add("World");//Store String
a1.add(new Integer(18));//Compile error… expects only Strings!!
```


`Vector` methods (among others) - Deprecated

```
Vector v = new Vector();
```

- **`isEmpty`**`(): returns true if vector is empty (false otherwise)`

```
boolean empty = v.isEmpty();
```

- **`addElement`**`(Object o): adds new element to vector`

```
v.addElement(new Ponto(1, 3));
```

- **`removeElement`**`(int i): removes i-index element from vector`

```
v.removeElement(i);
```

- **`removeAllElements`**`(): removes all elements from vector`

```
v.removeAllElements());
```

- **`firstElement`**`(): returns first object stored in the vector`

```
Ponto p = (Ponto)v.firstElement();
```

- **`elementAt`**`(int i): returns i-index object stored in vector`

```
Ponto p = (Ponto)v.elementAt(i);
```

- **`size`**`(): returns number of elements stored in vector`

```
int size = v.size();
...
```

`ArrayList` methods (among others)

`ArrayList<Ponto> alist = new ArrayList();`

- **`isEmpty`**`(): returns true if list is empty (false otherwise)`

```
boolean empty = alist.isEmpty();
```

- **`add`**`(E e): adds a <Generic> element to the list`

```
alist.add(new Ponto(1, 3));
```

- **`get`**`(int index): removes index element from list`

```
alist.removeElement(i);
```

- **`remove`**`(int index): removes element from index position in list`

```
Ponto p = (Ponto)alist.remove(0);
```

- **`remove`**`(E e): removes first occurrence of element in list`

```
boolean exists = alist.remove(p);
```

- **`set`**`(int index, E e): replaces element in index position by new e`

```
Ponto p = alist.set(0, p);
```

- **`clear`**`(): removes all elements from list`

```
alist.clear());
```

- **`contains`**`(Object o): cheks if list contains object`

```
boolean exists = alist.contains(p);
```

- **`indexOf`**`(Object o): returns índex of 1st occurrence of object in list`

```
int index = alist.size();
```

- **`size`**`(): returns number of elements stored in list`

```
int size = alist.size();
...
```


Exercises

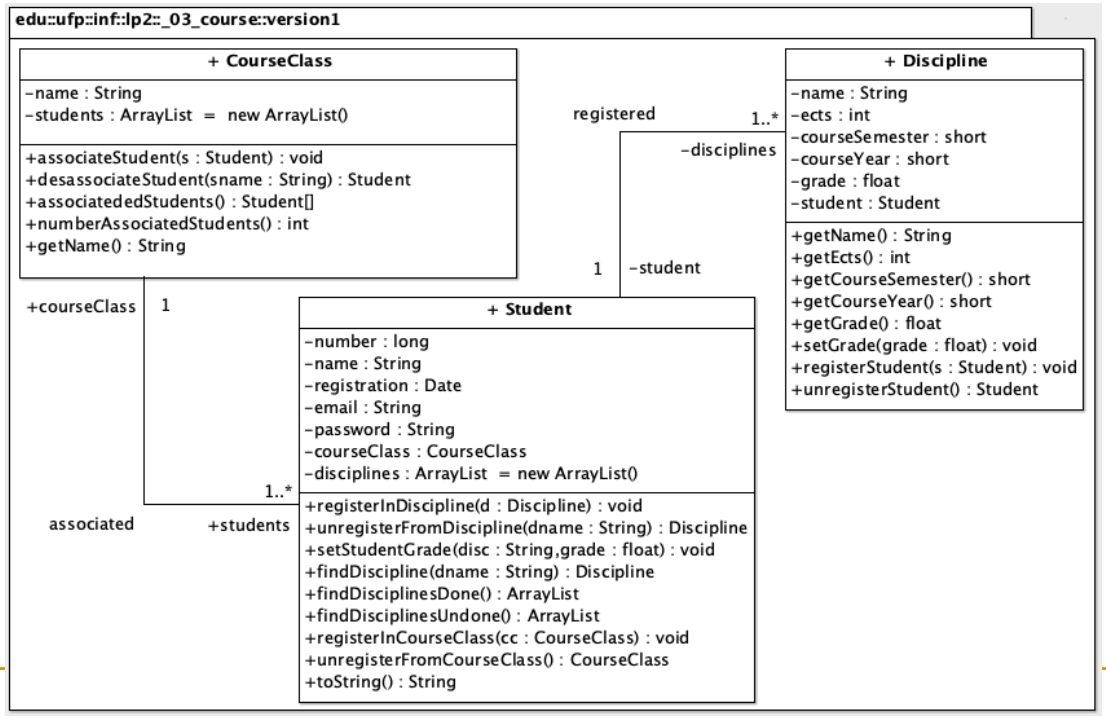
- Create `MyPrimitiveArrayList` class with `innerArrayList` for managing primitive types:

```
public class MyPrimitiveArrayList {
   private ArrayList innerArrayList = new ArrayList();

   // Add primitive type elements to list
   public void addInt(int i): stores Integer
   public void addFloat(float f): stores Float
   public void addChar(char c): stores internally Character
   public void addBoolean(boolean b): stores Boolean
   public int indexOf(Object o): get first index of object stored
   public int lastIndexOf(Object o): get last index of object

   // Get primitive type elements from list
   public int getIntAt(int index): returns int element
   public float getFloatAt(int index): returns float element
   public char getCharAt(int index): returns char element
   public boolean getBooleanAt(int index): returns boolean element
}
```

Exercises

- Create `_03_course.version1` package with classes:

  `CourseClass 1<->1..* Student 1<->1..* Discipline`



Exercises

- Create `_03_course.version2` package by including the `Grade` association class between `Student` and `Discipline` (tackle inconsistency problems):

  `Student 1<->1..* Grade 1..*<->1 Discipline`