CS133: Developing Programming Principles

Lecture 12

Arrays as parameters, arrays as return types, partially filled arrays

Example: working with arrays

- Develop a class for a list of birthdays.
- Use array of **`Date`** objects as instance variable for class.

```
public class BirthdayList {
  // declare array
  private Date[] birthday;
  //more code
}
```

Example continued

```
// create BirthdayList object
public BirthdayList(int numFriends)
{
  this.birthday = new Date[numFriends];
  Scanner in = new Scanner(System.in);
  for (int i = 0; i < this.birthday.length; i++)
  {
    this.birthday[i] =
        new Date(in.nextInt(), in.nextInt(),
                 in.nextInt());
  }
}
```


Passing an array as a parameter

- Declare array in parameter list in "usual way":
  `public` ***ReturnType MethodName*****(*****BaseType*****[] arrayName)**
- BaseType can be primitive or class name
- Use array in method in "usual way"

Arrays as parameters

- Write a method in the **`BirthdayList`** class that prints out birthdays of a selection of friends.
- The birthdays to print will be passed to the method as an array of integers.
- Each integer corresponds to the position in the array.

Solution

```
public void printSelection(int[] whichOne)
{
  for(int i = 0; i < whichOne.length; i++)
  {
    System.out.println(this.birthday[whichOne[i]]);
  }
}
```

Using the **`printSelection`** method

```
BirthDayList myDB = new BirthdayList(4);
    // Input 12 ints
int[] subset = new int[2];
subset[0] = 0;
subset[1] = 3;
myDB.printSelection(subset);
```


Arrays as return types

- Write a method **`createSelection`**
- Works like **`printSelection`**, but creates an array with the selected entries and returns it.

Returning an array from a method

- Create an array of the appropriate size
- Initialize all array entries for the new array
- Use return statement
- Method return type should be ***BaseType***[ ]

Arrays as return types continued

```
public Date[] createSelection(int[] whichOne)
{
  Date[] smallerArray = new Date[whichOne.length];
  for (int i = 0; i < whichOne.length; i++)
  {
    smallerArray[i] = this.birthday[whichOne[i]];
  }
  return smallerArray;
}
```

Arrays as return types – memory model

Challenge

- Write a method

```
public Date[] allBirthdaysInMonth(int m)
```

that returns an array of all entries in original list for the given month.

Partially filled arrays

- An array does not need to have all of its positions filled in order to be used.
- By default, when an array is created, each entry is filled with the default value for the base type:

Type	a[1]
`String[] a = new String[5];`	`null // all objects`
`int[] a = new int[5];`	`0 // all numeric primitives`
`boolean[] a = new boolean[5];`	`false`
`char[] a = new char[5];`	`'' // ASCII character 0`

Why have partially filled arrays?

- Example:
  - An array that represents the number of workers in a factory.
  - The factory can employ up to 100 people.
  - Currently employs 72.
  - Array needs to have space for the potential 28 workers.

Keeping track of partially filled arrays

- Use a counter variable to know how many entries are filled.
- Increment counter when entry is added.
- Use counter instead of length of array when iterating.

Example

- Suppose we want to have a calculator memory.
- It can hold up to 10 values
- Values should be doubles
- Methods should be:
  `clearMemory()` – all 10 cells are set to 0.0
  `addToMemory(double value)` – add value in next position, if valid.
  `maxMemory()` – returns the maximum value
  `sumMemory()` – returns the sum of elements

Example continued

```
public class CalcMem {
  public static final int MEM_SIZE = 10;
  private double[] memory =
            new double[MEM_SIZE];
  private int numElements = 0;

  public void clearMemory() {…}
  public void addToMemory(double value) {…}
  public double maxMemory() {…}
  public double sumMemory() {…}
}
```


Sequential search

- The simplest way to search for a value in an array.
- Start at index 0 and keep going through elements one by one until entry is found or end of array is reached.

Deleting array elements

- Deleting last element: relatively easy
- Deleting middle elements: trickier?

```
private Date[] bdays;
private int numBdays;
private void arrayDelete(int pos)
{
   // How to implement?
}
```


Growing arrays

- What should we do when we try to add to a full partially filled array?
  - Generate an error?
  - Grow the array?
- Idea: Make a new array of double the size and copy the elements over.

Why should we double the size?

ERROR: undefined
OFFENDING COMMAND:

STACK: