CSC 142

Arrays

[Reading: chapter 12]

Motivation

- Change the grades of a class from a 0-100 scale to a 0-4 scale.

```
double grade1, grade2, grade3;
// input the grades
...
// Change the scale
grade1*=4./100.;
grade2*=4./100.;
grade3*=4./100.;
/* Tedious! What if there are 100
 grades? */
```


Declaring an array

- A named collection of data items of the same type.
- To declare an array of String, write

```
String[] myText;
```

- To declare an array of integers, write

```
int[] intArray;
```

- The name of an array is an identifier
  - it follows the usual rules (letters or digits…)
- At this point, no memory is reserved for the array

Initializing an array

- In Java, an array is an object
- To initialize an array (once declared)

```
myText=new String[5];//5 Strings in myText
intArray=new int[8]; //8 ints in intArray
```

- Elements of an array are automatically initialized (0 for numbers, false for booleans, '\0' for chars, null for objects)
- The size of an array must be a non negative integer. It can be an integer variable.

```
int n = input.readInt("number of colors");
Color[] colors = new Color[n];
// can combine declaration and initialization
// on the same line
```

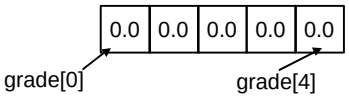

Array elements

- To access an array element, use its index (also called subscript), e.g.,

```
int[] intArray = new int[8];
```

  - intArray[3] refers to the element of intArray with index 3
  - The first element of an array has index 0
  - intArray[3] is the 4th element of intArray
- Memory view,

```
double[] grade = new double[5];
```



Using array elements (1)

- Array elements can be used anywhere a regular variable of the same type can be used.
- Example: array of primitives

```
boolean[] flags=new boolean[3];
flags[0]=true;
flags[2]=foo();//foo returns a boolean
/* OK? Is flags[1] initialized? */
System.out.print("flags[1]="+flags[1]);
if (flags[0])
 System.out.print("flags[0] is true");
```

Using array elements (2)

- Example: array of objects

```
Date[] milestones=new Date[10];
// man walks on the Moon
milestones[0]=new Date(1969,7,21);
// user's date of birth
//dateOfBirth returns a Date
milestones[1]=dateOfBirth();
// initialize the rest of milestones
// elements of index 2 to 9
//...
if (milestones[2].equals(new Date()))
 System.out.print("today is a milestone");
```


Array element initialization

- Arrays are automatically initialized (0 for numbers, false for booleans, '\0' for chars, null for objects)

```
String[] x = new String[6];
// x[0],…,x[5] are null
```

- When creating an array, one can initialize it with some initial values

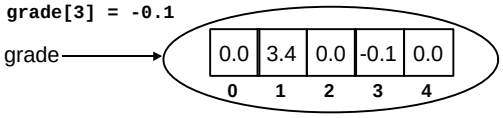
```
Color[] c={Color.blue,new Color(24,98,5)};
//works only on the declaration line
double[] t;
t={100.5,215.6,-23.3}; //Error
```

- Element by element

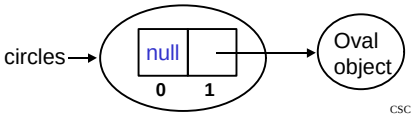
```
int[] a = new int[5];
a[0]=2; a[1]=3;
```


Memory view

```
double[] grade = new double[5];
grade[1] = 3.4;
grade[3] = -0.1
```




```
Oval[] circles = new Oval[2];
circles[1] = new Oval();
```



Example

- print the name of the month given its number

```
public void printMonth(int month)
{
  String[] names=
   {"January","February","March",
    "April","May","June","July","August",
    "September","October","November",
    "December"};
   System.out.print(names[month-1]);
   // Why –1?
}
```


Array size

- In java, the number of elements of an array is stored in the array object (in the public final field named length)
- e.g., an array of size 5

```
double[] x=new double[5];
System.out.print(x.length); // prints 5
```

- For an array of size N, it is an error to use an index outside the range 0,…, N-1, e.g.

```
x[x.length]=3; //Error
               //index should be between
               // 0 and x.length-1
```

- Array of size 0, e.g.
  `String[] text = new String[0];`
  text has no element (but is not null)

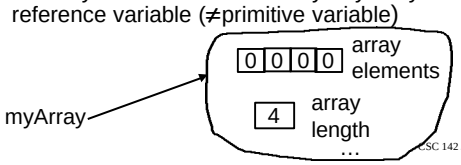
Working with arrays

- Iterate through an array using a loop

```
double[] grades; Input input=new Input();
// Size of the array
int n = input.readInt("How many grades?");
grades = new double[n];
// Input the grades
for(int i=0; i<grades.length; i++)
  grades[i]=input.readDouble("Enter grade");
// Compute the average
double average=0;
for(int i=0; i<grades.length; i++)
  average+=grades[i];
average/=grades.length;
```

Arrays and memory

- When creating an array, e.g.,
  ```
  int myArray[] = new int[4];
  ```
  - Java allocates memory to store the elements of the array
  - Java allocates memory to store information about the array (e.g. the length of the array)
- The name, myArray, refers to the block of memory that contains the array. myArray is a reference variable (≠primitive variable)



Recall: Primitive ≠ Reference

- A primitive type variable name refers to the value of the variable, e.g.
  ```
  int i = 3;  // i is 3
  int j = i;  // i and j are 3
  j = 4;      // i is 3, j is 4
  ```
- A reference type variable name refers to the memory location of the variable, e.g.
  ```
  int[] a = new int[5];
  int[] b = a; /*b refers to the same
   array as a */
  b[2]=3; //b[2] and a[2] are 3
  ```

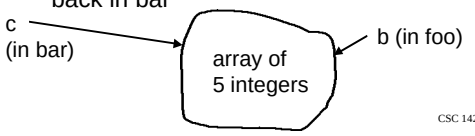

Arrays and methods

- An array is an object. Use the usual syntax to pass an array to a method.

```
public void foo(int[] b)
{
  for(int i=0; i<b.length; i++) b[i]=i;
}
//in some other method bar (of the same class)
public void bar()
{
  int[] c = new int[5];
  foo(c); // send c to foo
  for(int i=1; i<c.length; i++)
    System.out.println("c["+i+"]="+c[i]);
  // What is printed?
}
```


What is going on?

- foo(c);
  - call foo with c as an actual parameter
  - The value of c is copied in the formal parameter b (OK, since b and c have the same type, namely int[])
  - But the value of c refers to the location of an array of integers in the memory. Thus, b refers to the same memory location.
  - Any change of the array in foo is seen when back in bar



Why using references for arrays?

- An array can use a lot of memory
- Making a copy of the array could be prohibitive in terms of time and memory
- References allow us to modify variables in a method and pass the changes back.