CS108 Discussion Section

Java Arrays, Packages, and Modifiers

Orr Keshet

orr@cs

Discussion Section

- TA:

  Orr Keshet

  orr@cs

- Time & Location:

  Fridays 3:15-4:05pm

  Gates B01

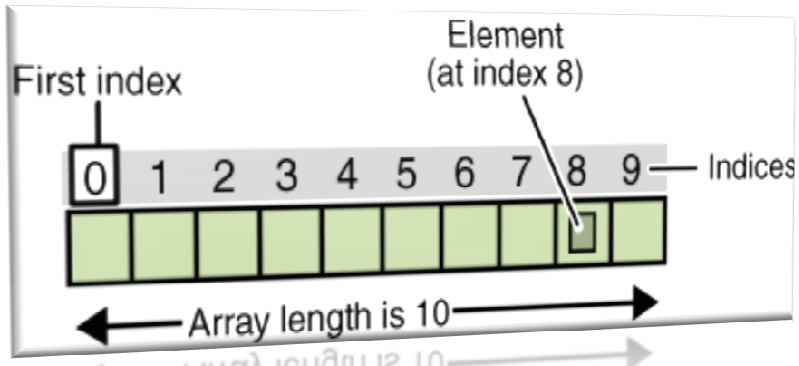
  Sections will not be held every week. See calendar on course website and email announcements.

Topics

- Arrays
  - Declaring Arrays
  - Initializing Arrays
  - Alternative Syntax
  - Accessing an Array
  - Multidimensional Arrays
- Packages
  - Setting a Package
  - File System relation to Packages
  - Using Package Members
- Modifiers
  - Access Modifiers
  - Static
  - Final

Arrays

- An array is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created. After creation, its length is fixed.
- All data types can be put into arrays



Credit: Sun Microsystems

Declaring Arrays

Java

- Declaration and allocation in two statements
  - `int[] students;`
  - `students = new int[100];`
- Single statement
  - `int[] students = new int[100];`
- Using an existing array
  - `int[] gradStudents = new int[100]`
  - `int[] students = gradStudents ;`

C++

- Declaration syntax
  - `int students[10];`

Initializing Arrays

- Once an array is allocated all initial values are 0 for numbers, false for booleans, null for references

- Examples

```
int[] students = new int[5];
System.out.println(students[3]);
```

Output?

```
String[] students = new String[2];
System.out.println(students[1]);
```

Output?

Initializing Arrays

- Once an array is allocated all initial values are 0 for numbers, false for booleans, null for references

- Examples

```
int[] students = new int[5];
System.out.println(students[3]);
```

0

```
String[] students = new String[2];
System.out.println(students[1]);
```

Output?

Initializing Arrays

- Once an array is allocated all initial values are 0 for numbers, false for booleans, null for references

- Examples

```
int[] students = new int[5];
System.out.println(students[3]);
```

0

```
String[] students = new String[2];
System.out.println(students[1]);
```

null

Alternative Syntax

- `String[] staff = {“Red”, “Sean”, “Patrick”, “Orr”};`
- `int[] numbers = {4,2,1};`

How to find array length?

- Examples using .length
  - `System.out.println(staff.length);`

Output?

  - `System.out.println(numbers.length);`

Output?

Alternative Syntax

- `String[] staff = {“Red”, “Sean”, “Patrick”, “Orr”};`
- `int[] numbers = {4,2,1};`

How to find array length?

- Examples using .length
  - `System.out.println(staff.length);`

4

  - `System.out.println(numbers.length);`

Output?

Alternative Syntax

- `String[] staff = {“Red”, “Sean”, “Patrick”, “Orr”};`
- `int[] numbers = {4,2,1};`

How to find array length?

- Examples using .length
  - `System.out.println(staff.length);`

4

  - `System.out.println(numbers.length);`

3

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java

- `a[2]`
- `a[a.length - 1]`
- `a[15]`

C++

- `int length =`
  `sizeof(a) / sizeof(int)`
- `a[2]`
- `a[length - 1]`
- `a[15]`

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java

- `a[2]`
- `a[a.length - 1]`
- `a[15]`

C++

- `int length = sizeof(a) / sizeof(int)`
- `a[2]`
- `a[length - 1]`
- `a[15]`

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java		C++
		`int length =` sizeof(a) / sizeof(int)
`a[2]`	=	`a[2]` 7
`a[a.length - 1]`		`a[length - 1]`
`a[15]`		`a[15]`

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java		C++	
		`int length = sizeof(a) / sizeof(int)`	
`a[2]`	=	`a[2]`	7
`a[a.length - 1]`	=	`a[length - 1]`	9
`a[15]`		`a[15]`	

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java

- `a[2]` =
- `a[a.length - 1]` =
- `a[15]`

`java.lang.ArrayIndexOutOfBoundsException`

C++

- `int length = sizeof(a) / sizeof(int)`
- `a[2]` 7
- `a[length - 1]` 9
- `a[15]`

Segmentation fault or unexpected behavior

Accessing an array

Consider: `int[] a = {3, 5, 7, 9};`

Java

- `a[2]` =
- `a[a.length - 1]` =
- `a[15]`

`java.lang.ArrayIndexOutOfBoundsException`

- Java checks bounds at runtime

C++

- `int length = sizeof(a) / sizeof(int)`
- `a[2]` 7
- `a[length - 1]` 9
- `a[15]`

Segmentation fault or unexpected behavior

- C++ doesn't check bounds for you

Multidimensional Arrays

- In Java, a multidimensional array is simply an array whose components are themselves arrays

- `int[][] grid = new int[2][4];`
- `int[] array = grid[0]`

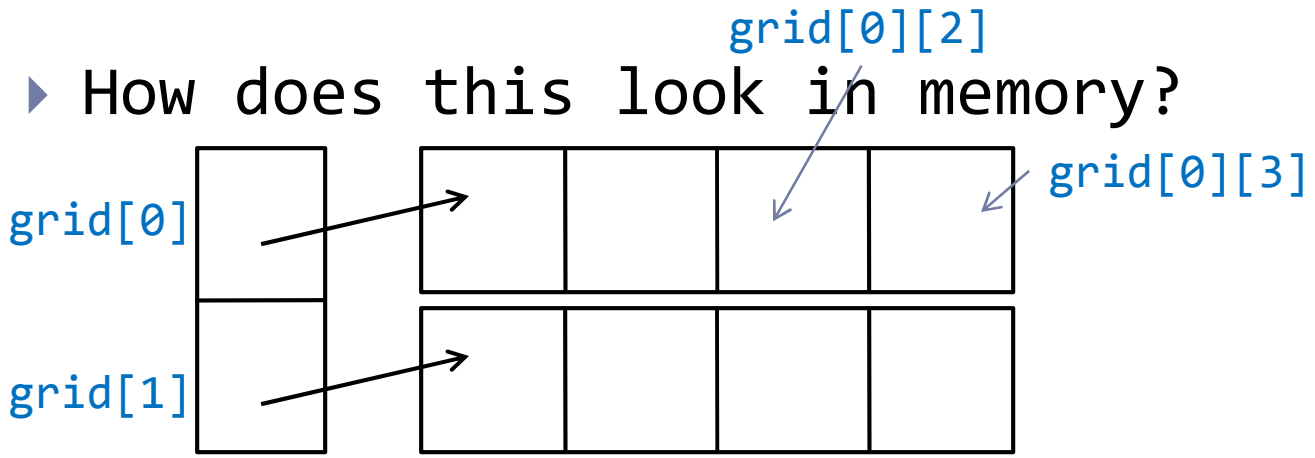
- How does this look in memory?

Multidimensional Arrays

- In Java, a multidimensional array is simply an array whose components are themselves arrays

- `int[][] grid = new int[2][4];`
- `int[] array = grid[0]`

- How does this look in memory?



Multidimensional Arrays

- Length example:

```
int[][] grid = new int[11][24];
int[] array = grid[0]

System.out.println(grid.length)

System.out.println(grid[0].length)

System.out.println(array.length)
```

Output?

Output?

Output?

Multidimensional Arrays

- Length example:

```
int[][] grid = new int[11][24];
int[] array = grid[0]

System.out.println(grid.length)
System.out.println(grid[0].length)
System.out.println(array.length)
```

11

Output?

Output?

Multidimensional Arrays

- Length example:

```
int[][] grid = new int[11][24];
int[] array = grid[0]

System.out.println(grid.length)
System.out.println(grid[0].length)
System.out.println(array.length)
```

11

24

Output?

Multidimensional Arrays

- Length example:

```
int[][] grid = new int[11][24];
int[] array = grid[0]

System.out.println(grid.length)
System.out.println(grid[0].length)
System.out.println(array.length)
```

11

24

24

Questions about Java arrays?

Packages – Why do we need them

- To make classes and interfaces easier to find and use
- to avoid naming conflicts
- to control access

Packages

- A *package* is a grouping of related classes or interfaces providing access protection and name space management.

- The package statement must be the first line in the source file

- Example:

```
package tools;
package graphics;
```

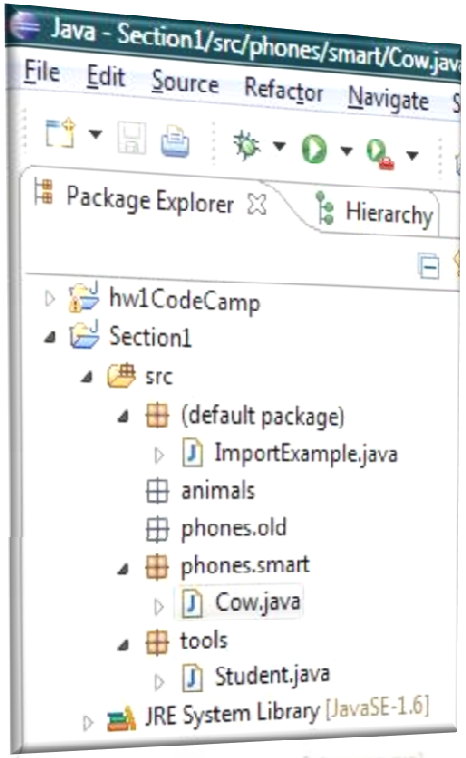
- If you do not use a package statement, your class or interface ends up in an unnamed package (also called default package)

Packages and the file system

- You can think of packages as directories in your file system
- Source file must be in the appropriate directory

- Example:

(switch to Eclipse)



Using package members

- Three ways to access a member that is not in your package:
  - Refer to the member by its fully qualified name
    - `java.util.ArrayList<String> list;`
  - Import the package member
    - `import java.util.ArrayList;`
      `ArrayList<String> list;`
  - Import the member's entire package
    - `import java.util.*;`
      `ArrayList<String> list;`

Note: import statement of the form: import java.util.*; does not import subdirectories. (go to DEMO)

Questions about Java Packages?

Java Modifiers

- There are various modifiers in Java, here are some of them (partial list)
  - abstract
  - final
  - private
  - protected
  - public
  - static
  - synchronized

Java Modifiers

- There are various modifiers in Java, here are some of them (partial list)
  - abstract
  - final
  - private
  - protected
  - public
  - static
  - synchronized

Covered Today

Access Modifiers

- public, private, protected, (default)
- Access level modifiers determine whether other classes can use a particular variable or invoke a particular method
- Two types of access control
  - Top Level
  - Member Level

Top Level Access Control

public

```
public class Cow {
    …
}
```

- Makes Cow visible to all other classes

  (still may need to import)

default

```
class Cow {
…
  }
```

- Cow is only visible within its package

Member Level Access Control

- Refers to methods and variables within the top level class

- Consider the following method

```
<modifier> int returnOne() {
    return 1;
}
```

- public => accessible to all
- private => accessible within its class
- protected => accessible within package and by subclasses
- no modifier => accessible within package only

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none				
public				
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No			
public				
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes		
public				
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes	No	
public				
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes	No	Yes
public				
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes	No	Yes
public	Yes	Yes	Yes	Yes
private				
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes	No	Yes
public	Yes	Yes	Yes	Yes
private	No	Yes	No	No
protected				

Summary of Access Modifiers

Modifier	Anyone	Within Class	Subclasses	Within Package
none	No	Yes	No	Yes
public	Yes	Yes	Yes	Yes
private	No	Yes	No	No
protected	No	Yes	Yes	Yes

(go to demo)

Static Modifier

- use the static modifier to create variables and methods that belong to the class rather than to a specific instance of it

- Example:

```
public class Animal {
  public static int numAnimals = 0;
  public Animal() {
    numAnimals++;
  }
}
```

Class variable example

```
public class Lion {
   public static int numAnimals = 0;
   public Lion() {
      numAnimals++;
   }
}
```

- What is the output of this program?

```
Lion lion1 = new Lion();
Lion lion2 = new Lion();
Lion lion3 = new Lion();
System.out.println(Lion.numAnimals);
```

Output?

Static variable example

```
public class Lion {
   public static int numAnimals = 0;
   public Lion() {
      numAnimals++;
   }
}
```

- What is the output of this program?

```
Lion lion1 = new Lion();
Lion lion2 = new Lion();
Lion lion3 = new Lion();
System.out.println(Lion.numAnimals);
```

3

Another example

- java.lang.Math class variables and methods are designed for static access

- Examples:

```
double y = Math.abs(2.5); (class method)
double logy = Math.log(y); (class method)
double pi = Math.PI;       (class variable)
```

Final modifier

final class

- no subclasses of a final class

(important for security)

final variable

- value can be assigned once
- ```
  static final
  double PI =
  3.14159265358;
  ```

final method

- no method can overwrite a final method

(import for security and design integrity)

(Go To Demo)


  ```

Questions about Modifiers?