CS102

Introduction to data structures, algorithms, and object-oriented programming

DAY 9
OPTION 1:
The midterm on Monday will be open notes.

I have posted a practice exam and the solutions to the on-line midterm from lab 4 on the course web page.
Review of static vs non-static class members

From outside classes:

All public methods and data fields, both static and non-static (instance fields), of a class are accessible through an object created from a particular class.

Within one class:

Static (aka class) methods can access static data fields within the same class directly, but must access instance variables or instance methods through objects.

Instance methods can access all types of data fields and static methods.
Data Encapsulation

Sometimes referred to as data hiding, is the principle whereby the implementation details of a class are kept hidden from the user. The user can only perform a restricted set of operations on the private members of the class by executing public methods.
```csharp
class Program {
    public class Account {
        private decimal accountBalance = 500.00m;

        public decimal CheckBalance() {
            return accountBalance;
        }
    }

    public static void main(String[] args) {
        Account myAccount = new Account();
        decimal myBalance = myAccount.CheckBalance();

        /* This main method can check the balance via the public
         * "CheckBalance" method provided by the "Account" class
         * but it cannot manipulate the value of "accountBalance" */
    }
}
```
Passing data into methods

Primitive types:
• passed into methods using a "pass-by-value" scheme.
• local copies of primitive parameters are created, even if the parameters and arguments have the same name.

Object types:
• passed into methods using "pass-by-value".
• actually passing in only the address (a reference) to the object.
• changes made to the object inside (local to) the method will also be made to the object in the larger scope in which the call is made.
class Octopus {

    // instance data field
    private int numLegs;

    // class (static) data field
    private static final int MAX_LEGS = 8

    // note use of this keyword in next 3 methods
    public Octopus(int numLegs) {
        this.numLegs = numLegs;
    }

    public void setNumLegs(int numLegs) {
        this.numLegs = numLegs;
    }

    public int getNumLegs() {
        return this.numLegs;
    }
}

Passing data into methods

Ex: Suppose we had methods (2) through (5) declared in the Octopus class:

(2) public static void main(String[] args) {
    Octopus octy = new Octopus(MAX_LEGS);
    // sending an object into a method
    removeLegs(octy);
    Octopus octet;
    // returning an object from a method
    octet = mutantOctopus();
}

(3) public static void removeLegs(Octopus octo) {
    octo.setNumLegs(octo.getNumLegs() - 1);
}

(4) public static Octopus mutantOctopus() {
    Octopus newMutant = new Octopus(getRandomLegs());
}

(5) public static int getRandomLegs() {
    java.util.Random generator = new java.util.Random();
    return 1 + generator.nextInt(MAX_LEGS);
}
Objects can be created inside methods and returned just like primitive types. For example, the mutantOctopus method (4) has return type Octopus. An Octopus object is created inside the mutantOctopus method and "newMutant" is the local name for this object. When control is returned to the main program, the Octopus object reference "octet" is set to point at the returned object.
Arrays of Objects

Person[] people = new Person[10];

for (int i = 0; i < people.length; i++) {
    String name = javax.swing.JOptionPane.showInputDialog(null, "Enter name "+i+": ");
    int age =
        Integer.parseInt(javax.swing.JOptionPane.showInputDialog(null, "Enter age of "+i+": "));

    // This is the step most often forgotten -
    // creating each individual object in the array.

    people[i] = new Person(name, age, gender);
}
Arrays of objects

IMPORTANT: Every time you create an array of objects you must use the new operator once for the array declaration and once for each object in the array.

E.g., if the array length is 10, this means the new operator would be used 11 times to create all the objects in the array.