Advanced Objects

Class constructors – my favorite shortcut

- A CONSTRUCTOR is a shortcut method that fills values into an INSTANCE all in ONE line of code
- called as soon as you create an instance, ***AND ONLY THEN!!!***
- name of constructor method MATCHES the class it belongs to
- instance's data ***is empty*** until we set each member variable
- we need to CREATE the constructors
  - default
  - complete programmer defined
  - partial programmer defined
  - uses overloading, since all have the same name
- constructors are placed INSIDE the class they belong to

Constructors are our friends

Without Constructors

```
Student S0001 = new Student();
S0001.setTest1(100);
S0001.setTest2(100);
S0001.setTest3(100);

S0001.Test1 = 85;
S0001.name.equals("Lupoli");
```

```
EM001.setTitle("Assistant Professor");
EM001.setSalary(-1);
System.out.println(EM001); // uses toString method in class
Employee EM001 = new Employee();
EM001.setName("Mr. Lupoli");
EM001.setDepartment("Computer Science");
```

With Constructors

```
Student S0001 = new Student(100,100,100, "Lupoli");

Employee EM001 = new Employee("Mr. Lupoli","Computer Science","Assistant Professor",-1);
System.out.println(EM001); // uses toString method in class
```

Class Default Constructors

- A method INSIDE the class
- default constructor ALREADY created
  - sets values to a DEFAULT value called autoinitialization

Autoinitialization Chart	
Data Type	**Initial Value**
byte	0
short	0
int	0
long	0
double	0.0
double	0.0
char	space
Boolean	false
object reference	null
String	null

- called as soon as you create an instance, ***AND ONLY THEN!!!***
  - Employee EM002 = new Employee();
- you may create your own default constructor
- name of constructor method MATCHES the class it belongs to
- also notice no parameters ( )

Example Default Constructor

Code in class

```
Student( ) // this code goes into Student.java
{
     test1 = -1;
     test2 = -1;
     test3 = -1;
     average = -1;
}
```

Code to USE constructor

```
// in main()
Student Hardy = new Student();

System.out.println(Hardy); //would show all -1, using a
toString
```

// Create a SIMPLE default constructor for Employee Answer$_b$:

Complete Programmer Defined Constructors

- Programmer gets to place a value for EVERY single data member in the object
  - o notice parameters have values (x, y, z) for each data member

Example Complete Programmer Defined Constructor

Code in class

```
Student(int t1, int t2, int t3) // this goes into Student.java
{
     test1 = t1;
     test2 = t2;
     test3 = t3;
}
```

Code to USE constructor

```
Student Hardy = new Student(50, 75, 80);

System.out.println(Hardy); // would show all 50, 75, 80

// Complete a full constructor for Employee
```

Answer$_b$:

Partial Programmer Defined Constructors

- Programmer gets to place a value for SOME data members in the object

Example Partial Pro. Defined Constructor

Code in class

```
Student(int t1) // this goes into Student.java
{
     test1 = t1;
     test2 = -1;
     test3 = -1;
     average = -1;
}
```

Code to USE constructor

```
Student Hardy = new Student(50);

System.out.println(Hardy); // would show all 50, -1, -1
```

Final Word on Constructors

- you can have all three types of constructors in ONE program
  - o easy to figure out by counting the parameters
  - o but the name of the method will be the same since they use overloading

In main

```
Student Lupoli = new Student(100, 100, 100); // automatically fills
Student Kim = new Student (100, 90, 80);
Student Angela = new Student (88);
Student Chris = new Student (-100, -10, -80);
Student Andy = new Student();  // WILL CALL DEFAULT CONSTRUCTOR (OVERLOAD)
//Which constructor did it call??
```

#	

1	`Student( )`
	`{`
	`Test1 = 0;`
	`Test2 = 0;`
	`Test3 = 0;`
	`average = 0;`
	`}` // how many parameters does this constructor take?? Then which students above called this one?

2	`Student(int T1)`
	`{`
	`Test1 = T1;`
	`Test2 = 0;`
	`Test3 = 0;`
	`average = 0;`
	`}` // how many parameters does this constructor take?? Then which students above called this one?

3	`Student(int T1, int T2, int T3)`
	`{`
	`Test1 = T1;`
	`Test2 = T2;`
	`Test3 = T3;`
	`}` // how many parameters does this constructor take?? Then which students above called this one?

Completing an Advanced Class Profile

- before any of the Advanced Data Structure can be used to their full potential for CUSTOM data types
- The base class, no matter of what type must have shown below
  - interesting parts, compareTo, <, !=
- Questions to ask yourself
  - what is public/private accessible?
  - will individual instances be compared or sorted?

- The file order
  - Data members
    - Especially in Eclipse, since it can generate the code from the members
  - Constructors
  - Accessors
  - Mutators
  - Operators (compareTo/equals)
  - toString

Employee's Class Profile

```
import java.util.Scanner;

class Employee
{
        // data members
        private String firstname, lastname, title;
        private int age;
        public Scanner sc = new Scanner(System.in);

        // constructors
        Employee() {} // empty constructor, fill later

        Employee(String f, String l, String t, int a)
        // What other methods should we have??
        {
                firstname = f;
                lastname = l;
                title = t;
                age = a;
        }

        // accessors
        public String getfirstName(){ return firstname; }
        public String getlastName() { return lastname; }
        public int getAge()         { return age; }

        // mutators
        public void setfirstName(String f) { firstname = f; }
        public void setlastName(String f) { lastname = f; }
        public void setAge(int a) { age = a; }

      public Employee nextEmployee()
       {
                System.out.println("Enter first name");
                setfirstName(sc.next());
                System.out.println("Enter last name");
                setlastName(sc.next());
                System.out.println("Enter age");
                setAge(sc.nextInt());
                return this;
        }

        // operators
        public int compareTo(Employee x)
        {
                // same last name, then check first
                if (this.getlastName().compareTo(x.getlastName()) == 0)
                {
                        return this.getfirstName().compareTo(x.getfirstName());
                }
                else // last names were different
                {
                return this.getlastName().compareTo(x.getlastName());
                }
        }

        public boolean equals(Employee x)
        {
                if(         this.firstname.equals(x.firstname) &&
                                this.lastname.equals(x.lastname) &&
                                this.age == x.age)
                {return true; }
                else {return false; }
        }

        public String toString()
        { return getlastName() + ", " + getfirstName() + "\n " +
getAge(); }
}
```

1. Identify the data members
2. Find the constructors
3. How will the toString display the instance's data?
4. What does the compareTo function really compare?
5. What does the equals return and really compare?

Accessors versus Mutators

- functions within a class that access member variables
  - functions are EXTREMELY small+
- public functions, we need to use them in other classes
- accessors
  - "get" member values for instance
    - function names start with "get"
  - to NOT change member variable values
  - functions do not (usually) have parameter
- mutators
  - "set" or edit member values for instance
    - function names start with "set"
  - to change member variable values

Example Accessors and Mutators for Employee

```
// accessors
public String getfirstName()   { return firstname; }
public String getlastName()    { return lastname; }
public int getAge()      { return age; }

// mutators
public void setfirstName(String f) { firstname = f; }
public void setlastName(String l) { lastname = l; }
public void setAge(int a) { age = a; }
```

Review of the ToString Function

- overloads the String function “toString”
- created by the programmer
- used to display the instance and all of it’s values
- function is added to the class
- NOTICE no “”toString()” behind the instance!!
    - o called automatically when System.out.println(String) is called

ToString Function Example

Code

```
public String toString()
{
  return getfirstName() + ", " + getlastName() + "\n " + getAge();
}
```

Called

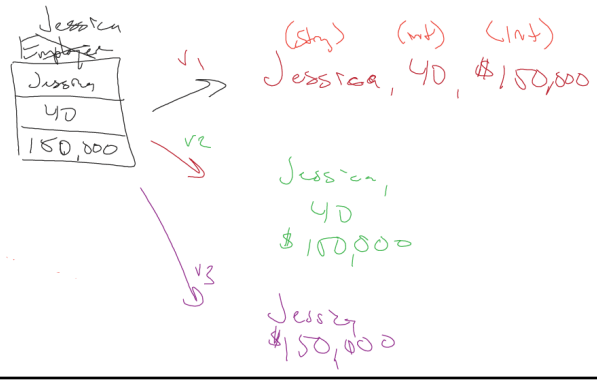
```
public class EmployeeDriver
{
      public static void main(String[] args)
      {
            Employee adjunct = new Employee("Shawn", "Lupoli", 21);
            Employee dean = new Employee("Jack", "McLaughlin", 75);
            Employee professor = new Employee("Super", "Mario", 81);

            System.out.println(adjunct); .// this requires toString to be prsent
      }

}
```

Output

```
Shawn Lupoli 21
```



Equals verses CompareTo

- both used originally for Strings
- equals
  - return true/false
- compareTo
  - replaces <, >!!!!
  - syntax
    - x.compare(y)
  - returned values
    - 0 == identical
    - > 0 == x and y are in reserve alphabetical order
    - < 0 == x and y are in alphabetical order

1. Copy and Paste Employee from example above into Eclipse
2. Create a new class EmployeeDriver.java that contains the main()
3. Create your own simple instance of Employee
4. Answer the question below
5. How is an Employee compared using the "compareTo" and "equal" functions?

Important String Comparing Tables

compareTo

Value returned	*Condition*
	a == b
	a < b
	a > b

Equals

Value returned	*condition*
	a == b
	a != b

Overloaded “equal” method

- again, overloads the String’s “equal” method
- compares each member-wise value
- created by the programmer

Overloading the “equals” operator

Function

```
public boolean equals(Employee x)
{
  if(   this.firstname.equals(x.firstname) &&
        this.lastname.equals(x.lastname) &&
        this.age == x.age)
  {return true; }
  else {return false; }
}
```

We will talk about “this” in a moment

Call

```
Employee dean = new Employee("Jack", "McLaughlin", 90);
Employee professor = new Employee("Peter", "Joyce", 81);
System.out.println(dean.getFirstName()); // would print what?

System.out.println(dean.equals(dean));
System.out.println(dean.equals(professor));
```

Result

```
true
false
```

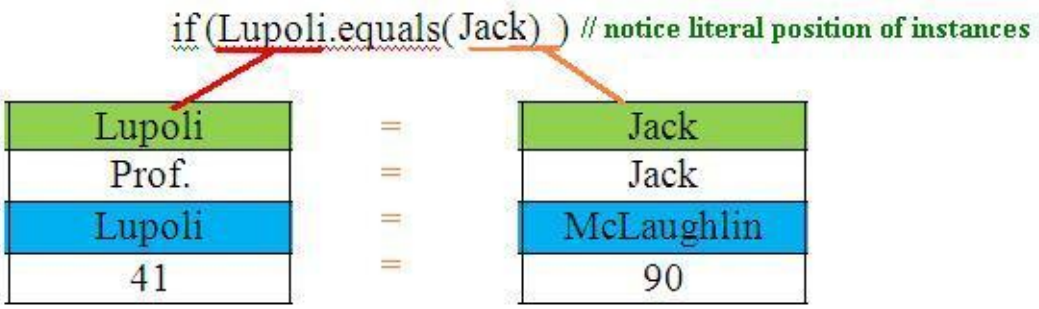
What is “this” again?

- really have to look at an example to explain
- used in compareTo/equals methods
- used for comparison (like below)

if (Lupoli.equals(Jack))

Explaining “this”. All about Position.

(instance name)	Jack	Lupoli	Jessie	Matt
firstname	Jack	Prof.	Jessie	Matt
lastname	McLaughlin	Lupoli	Orsburne	Farrow
age	90	41	19	21




```
this.firstname.equals(x.firstname) &&
this.lastname.equals(x.lastname) &&
this.age == x.age)
```

Lupoli Jack

```
this.firstname.equals(x.firstname) &&
this.lastname.equals(x.lastname) &&
this.age == x.age)
```

Who (really) is this?? Draw who is “this”??

if (Jack.equals(Jessie))

if (Lupoli.compareTo(Matt))

if (Matt.equals(Jack))

Overload the compareTo function

- again, overloads the String's "compareTo" method
- compares each member-wise value
- created by the programmer
- Have to ask yourself
  - What are we going to compare!!!
  - For Employee
    - Age (numeric)
    - Full name (string)

Overloading the CompareTo (numeric) operator

Function

```
public int compareTo(Employee x)
{
      if (this.age == x.age)
      { return 0; }
      else if (this.age < x.age)
      { return -1; }
      else // (this.age > x.age)
      { return 1; }
}
```

Overloading the CompareTo (String) operator

Function

```
public int compareTo(Employee x)
{ return this.getlastName().compareTo(x.getlastName()); }  // comparing STRINGS
```

Call

```
System.out.println(dean.compareTo(dean));
System.out.println(dean.compareTo(professor));
if(professor.compareTo(dean))
{

}
```

There is a problem with this CompareTo function String (in theory, about sorting names). What is it? Recreate the function. Answer$_b$:

So what does this look like overall?

- Let's put it all together
- Remember, two file system
  - Driver (has main)
  - Employee (class/object)

Putting it all together now

Class	Driver
Same as Employee (Complete Profile)	

```
public class Driver {

	public static void main(String[] args)
	{
		Employee adjunct = new Employee("Shawn", "Lupoli", 30);
		Employee dean = new Employee("Jack", "McLaughlin", 90);
		Employee professor = new Employee("Peter", "Joyce", 60);

		Employee Lupoli = new Employee();
		Lupoli.nextEmployee();
		System.out.println(Lupoli);
		System.out.println(Lupoli.toString());

		System.out.println(dean == dean);
		System.out.println(dean == professor);

		// Compare
		// x.compare(y)
		// 0 == identical
		// > 0 == x and y are in reserve alphabetical order
		// < 0 == x and y are in alphabetical order

		System.out.println(dean.compareTo(dean));
		System.out.println(dean.compareTo(professor));
		/*
			if(professor.compareTo(dean))
			{

			}
		 */

	}
}
```

Introduction to JOptionPane

- must import javax.swing.*;
- two types
  - input
  - message

Joption MESSAGE Types

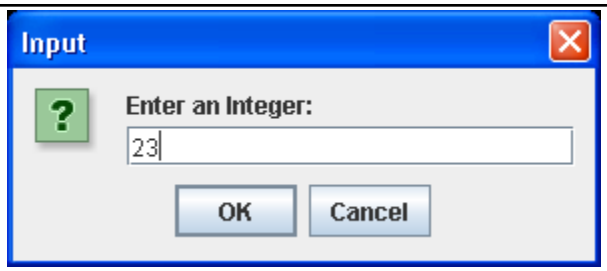
Code

```
JOptionPane.showMessageDialog (null, "Message", "Title", JOptionPane.INFORMATION_MESSAGE);
JOptionPane.showMessageDialog (null, "Message", "Title", JOptionPane.WARNING_MESSAGE);
JOptionPane.showMessageDialog (null, "Message", "Title", JOptionPane.ERROR_MESSAGE);
```

Information	Warning	Error
		

JOptionPanel Input

String s = JOptionPane.showInputDialog(null, “Enter an Integer:”);
System.out.println(“You entered “ + s); // converts String to Integer



Conversion you already have (String notes)

Overloading next*Something*()

- next() is a Scanner function used to enter data
- could use JOptionPane or Scanner to gather data
  - must import whatever library corresponds
- uses mutators to set values
  - don't reinvent the wheel

nextSomething Example

Function

```
public Employee nextEmployee()
{
        System.out.println("Enter first name");
        setfirstName(sc.next());
        System.out.println("Enter last name");
        setlastName(sc.next());
        System.out.println("Enter age");
        setAge(sc.nextInt());
        return this;
}
```

Call

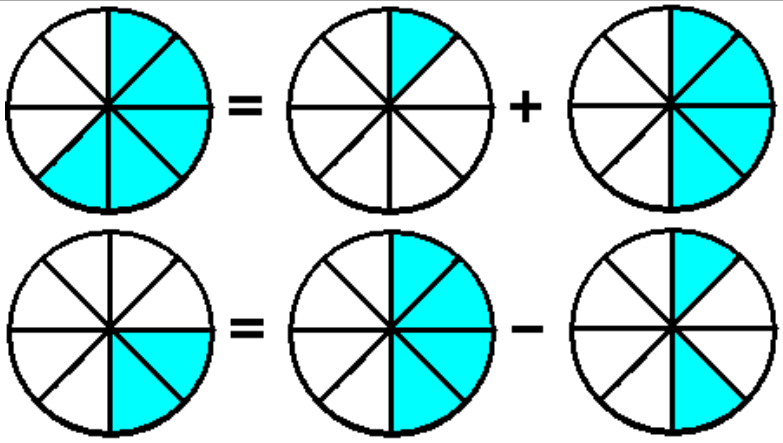
```
        Employee Lupoli = new Employee();
        Lupoli.nextEmployee();
        System.out.println(Lupoli);
```

-------- End of Advanced Classes Lab ------- (as of 4/12/16)

Adding Mathematical features

- NOT ALL INSTANCES REQUIRE MATHEMATICAL FEATURES!!
  - Employee sure doesn't
- C++ and other languages give the ability to ***overload operators*** such as +, -, etc…
- Java does not
- we can create and ***overload of functions***
  - we already have toString, compare, equal, etc…
  - we MAY need to create "add", "subtract", etc…

Examples of Using overloaded operators



Introduction to Math feature with PIE class

- the Pie class is a simple class with many of the complete profile functions
- we are focused on the mathematical features

Pie Class setup

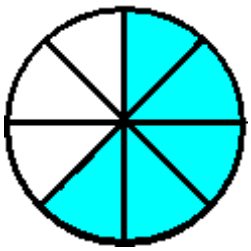
```
public class Pie
{
	private int pieces;
	private int MAX_PIECES;
	private String type;

	public Pie() //default pie
	{
		this.pieces = 8;
		this.MAX_PIECES = 8;
	}

	public Pie(String type) //default pie
	{
		this.type = type;
		this.pieces = 8;
		this.MAX_PIECES = 8;
	}

// complete class profile functions and features below

// scroll up to see what we need to have a complete class
// profile
```

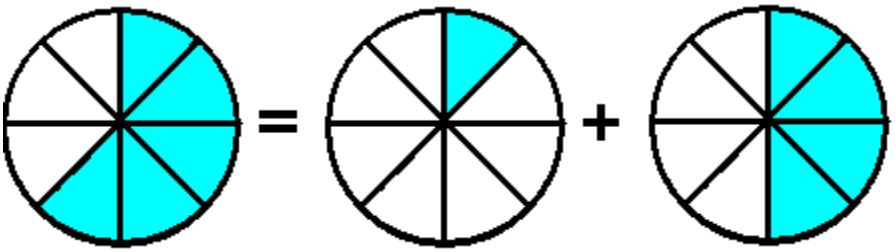


Addition “overload” method

- we call the function add
- add two pies to ONE pie
- will always need to identify exactly what we are adding together in two instances
  - in this case we are adding the member variable “pieces”
- make sure to add validation features

Adding Overload

Adding in theory



Adding function

```
public Pie add(Pie x)
{
	if(this.pieces + x.pieces > MAX_PIECES)
	{
		JOptionPane.showMessageDialog (null, "Too Full!! Cannot add to pie.", "Adding",
JOptionPane.ERROR_MESSAGE);
	}
	else if(!this.type.equals(x.type))
	{
		JOptionPane.showMessageDialog (null, "Wrong types!! Cannot add to pie.",
"Adding", JOptionPane.ERROR_MESSAGE);
	}
	else // there is room, and same type
	{
		this.pieces += x.pieces;
		x.pieces = 0;
	}
	return this;

}
```

Adding call

```
// combine into one pie!! WRONG TYPES!!
peachPie1.add(cherryPie2); // won't work with code above since types are different

// combine into one pie!!
cherryPie1.add(cherryPie2);
```

// Create the subtract method

Array of Objects

- NOT THE SAME AS OBJECTS WITH ARRAYS!!!!
- An array of Objects are exactly the same as an array of structs just again with variables and functions
- We can use an array of Objects just like an array!!
- We can use for loops to access a huge amount of data since the Objects are identified by indices!!!

```
Student [] CS1044 = new Student[100];
for(int i = 0; i < CS1044.length; i++)
{ CS1044[i] = new Student(); }
```

0	1	2	3	4	5	6	7	8
Test1	Test1	Test1	Test1	Test1	Test1	Test1	Test1	Test1
Test2	Test2	Test2	Test2	Test2	Test2	Test2	Test2	Test2
Test3	Test3	Test3	Test3	Test3	Test3	Test3	Test3	Test3

```
CS1044[0].Test1 = 70; // student #0 got a 70 on Test1
CS1044[1].Test1 = 40; // student #1 got a 40 on Test1
CS1044[2].Test1 = 90; // student #2 got a 90 on Test1
CS1044[4].getTestAverage( ); // will display the test average for student #4
```

0	1	2	3	4	5	6	7	8
70	40	90	Test1	100	Test1	Test1	Test1	Test1
Test2	Test2	Test2	Test2	100	Test2	Test2	Test2	Test2
Test3	Test3	Test3	Test3	100	Test3	Test3	Test3	Test3

```
for (int i = 0; i < 25; i++)
{ CS1044[i].getTestAverage( ); } // will display test averages for the entire Period1 class
```

```
CSIT211[0].fname = "Prof.";
CSIT211[0].lname = "Lupoli";
```

Displaying an ENTIRE array of Objects

REMEMBER!! It's just an array, with a CLASS inside!! Still acts like an array!!

So, how did we display a NORMAL array of 100 elements??

```
for (int i = 0; i < 100; i++) // this is how we did this with a NORMAL array
{ System.out.println(array[i]); }
```

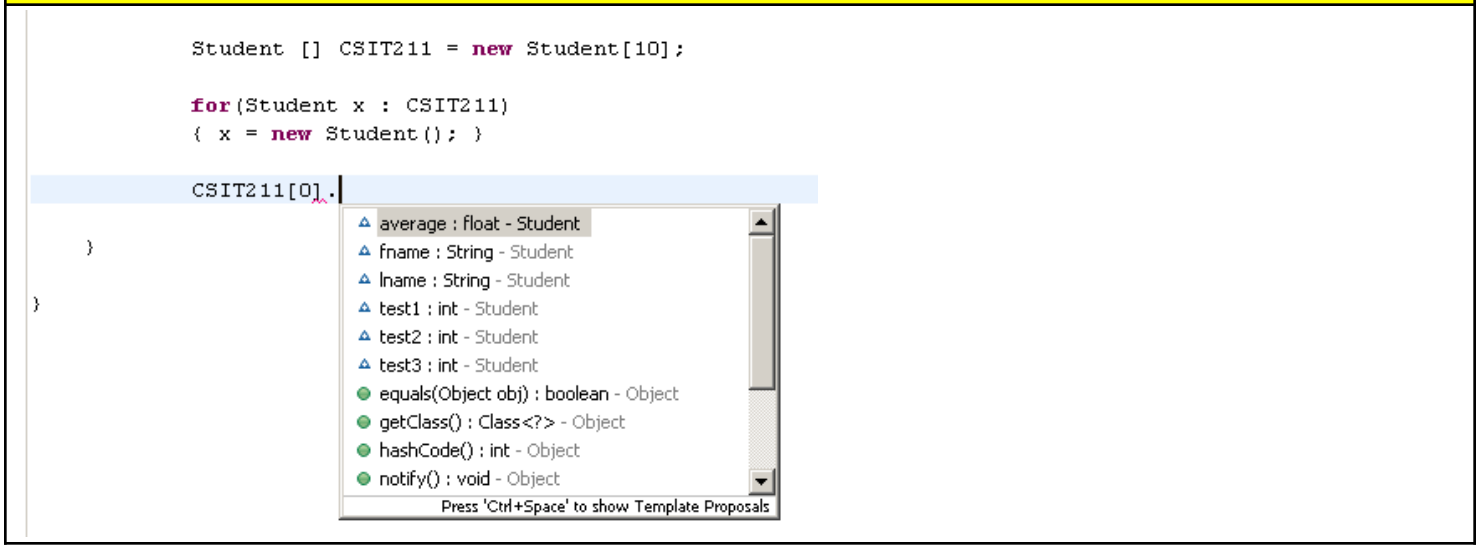
NOW WITH OUR STRUCTS, WE NEED TO DISPLAY EACH MEMBER VARIABLE!!

code WITHOUT toString	code WITH toString

for (int i = 0; i < 100; i++)
{
System.out.println(array[i].fname);
System.out.println(array[i].lname);
System.out.println(array[i].test1);
}
	for (int i = 0; i < 100; i++)
	{ System.out.println(array[i]); }

Let your IDE help you!!!

IDE adjusts to array to help you fill in data faster



Functions and Array of Objects

- can be tricky
- 2 possible scenarios
  - passing one element in the array of Objects
  - passing the ENTIRE array of Objects

in the main( )

```
Student [] Period1 = new Student [25];
// students are filled in

// I want to display ONE person's test average
Period1[3].getTestAverage( );  // NOTICE NO PARAMETERS!!!!
// What is being passed to the function???
```

the prototype/function (in Student.java)

```
public void getTestAverage( )
{
        average = float(Test1 + Test2 + Test3)/3;
}
```

Passing an ENTIRE array of Objects #1

in the main( )

```
Student [] Period1 = new Student [25];
// students are filled in

display_entire_getTestAverages ( Period1);
```

the prototype/function

```
public void display_entire_getTestAverages(Student []x)
{
        for(int i = 0; i < x.length; i++)
        {
        average = float(x[i].Test1 + x[i].Test2 + x[i].Test3)/3;
        System.out.println(average);
        }
}
```

Passing an ENTIRE array of Objects #2

in the main( )

```
Student [] Period1 = new Student [25];
// students are filled in
for(int i = 0; i < Period1.length); i++)
{ Period1[i].display_entire_ getTestAverages();  }
```

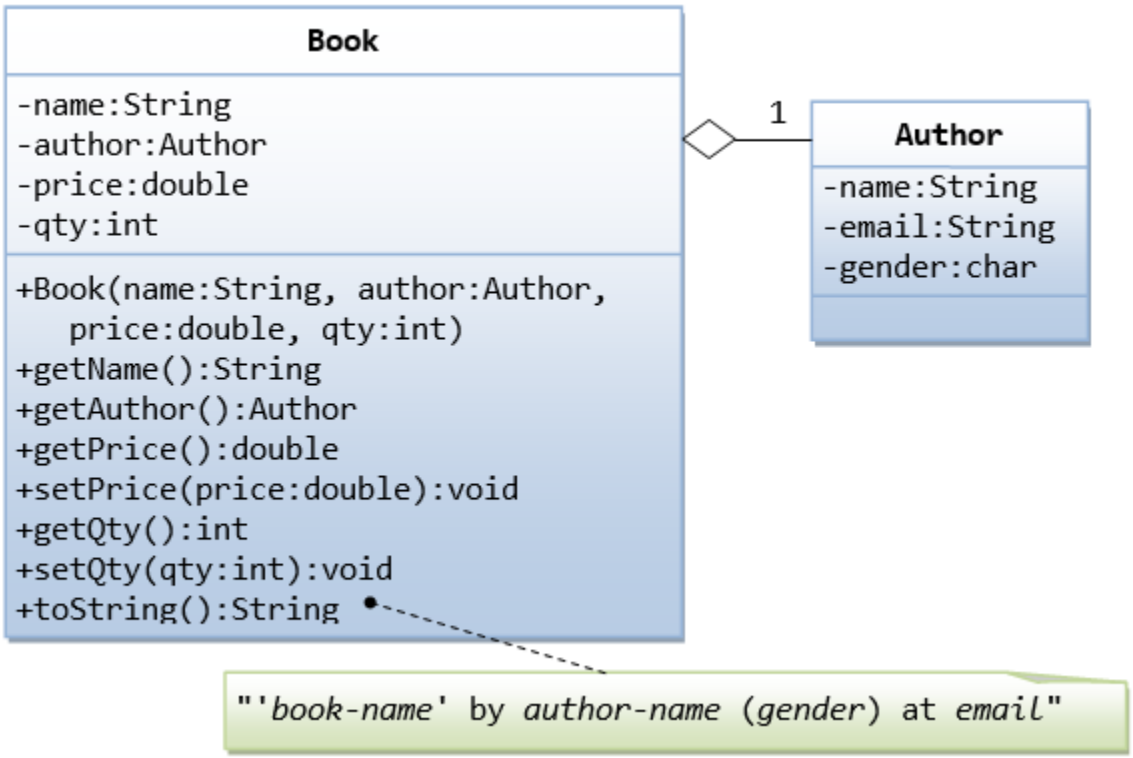
the prototype/function

```
public void display_entire_getTestAverages()
{
        average = float(Test1 + Test2 + Test3)/3;
        System.out.println(average);
}
```

Objects within Objects

- This is NOT inheritance!!
- Objects themselves can have other objects ***stored*** WITHIN them
  - code for the other object is still in another file
  - treat the inner object like a variable!!
- Create the inner class first (Author) so Eclipse can handle it while building the outer class (Book)

Objects within Objects



Using Objects with an Object (Person & Date)

Date Class

```
public class Date {

	private short month;
	short day;
	short year;

	public Date(short month, short day, short year) {
		this.month = month;
		this.day = day;
		this.year = year;
	}

	public short getMonth() {return month; }
	public short getDay() { return day; }
	public short getYear() { return year;}
	public void setMonth(short month) { this.month = month; }
	public void setDay(short day) { this.day = day;}
	public void setYear(short year) { this.year = year; }
	@Override
	public String toString() {
		return "Date [month=" + month + ", day=" + day + ", year=" + year + "]";
	}
}
```

Person Class

```
public class Person {
	String first;
	String last;
	Date date;

	protected Person(String first, String last, Date date)
	{
		this.first = first;
		this.last = last;
		this.date = date;
	}

	protected Person(String first, String last, short month, short day, short year)
	{
		this.first = first;
		this.last = last;
		this.date = new Date(month, day, year);
	}

	protected String getFirst() { return first; }
	protected String getLast() { return last; }
	protected Date getDate() { return date; }
	protected void setFirst(String first) { this.first = first; }
	protected void setLast(String last) { this.last = last; }
	protected void setDate(Date date) { this.date = date; }

	public String toString()
	{ return "Person [first=" + first + ", last=" + last + ", date=" + date + "]";}
}
```

Driver

```
public class Driver {

	public static void main(String[] args)
	{
		Date temp = new Date((short)12, (short)30, (short)1976);

		Person Richard = new Person("Richard", "Shaw", temp);
		System.out.println(Richard);

		Person Ashley = new Person("Ashley", "Pitt", (short)6, (short)6, (short)2001);
		System.out.println(Ashley);

		Person Justin = new Person("Justin", "Pain", new Date((short)2, (short)45, (short)1988));
		System.out.println(Richard);

		Justin.date.setDay((short) 23);
	}
}
```

Answers

Employee Default constructor

```
public Employee()
{
    this.name = null;
    this.department = null;
    this.title = null;
    this.salary = -1;
}
```

```
public Employee()
{
    name = "";
    department = null;
    title = null;
    salary = -1;
}
```

```
public class Employee {

	String name;
	String department;
	String title;
	int salary;

	public Employee()
	{
		this.name = null;
		this.department = null;
		this.title = null;
		this.salary = -1;
	}

	@Override
	public String toString() {
		return "Employee [name=" + name + ", department=" + department
				+ ", title=" + title + ", salary=" + salary + "]";
	}
	public String getName() { return name; }
	public String getDepartment() { return department; }
	public String getTitle() { return title; }
	public int getSalary() { return salary; }
	public void setName(String name) { this.name = name; }
	public void setDepartment(String department) { this.department = department; }
	public void setTitle(String title) { this.title = title; }
	public void setSalary(int salary) { this.salary = salary; }


}
```

```
		// using the default constructor
		Employee Ethan = new Employee();
		// let's see what he's got
		System.out.println(Ethan); // using the toString
```

Complete Programmer Defined Constructor

```
public class Employee {

	String name;
	String department;
	String title;
	int salary;

	public Employee()
	{
		this.name = null;
		this.department = null;
		this.title = null;
		this.salary = -1;
	}

	public Employee(String name, String department, String title, int salary) {
		this.name = name;
		this.department = department;
		this.title = title;
		this.salary = salary;
	}
```

```
		// using the complete programmer defined constructor
		Employee Melanie = new Employee("Melanie", "Grad School", "Director", 3000000);
		// let's see what he's got
		System.out.println(Melanie); // using the toString
```

Complete Employee compareTo (by Name)

```
if(return this.getlastName().compareTo(x.getlastName()) == 0) //same lastnames
{ return this.getFirstName().compareTo(x.getFirstName());  }
else // return lastname
{ return this.getlastName().compareTo(x.getlastName()); }
```

```
public class Driver {

	/**
	 * @param args
	 */
	public static void main(String[] args) {

		// set up variables
```

```
		double num1 = 10;
		double num2 = 20;
		double num3 = 30;

		double answer = getAverage(num1, num2, num3);

		System.out.println(answer);

	}

	public static double getAverage(double a, double b, double c)
	{
		return (a + b + c) / 3;
	}


}
```