SENG1110/SENG6110
Object Oriented Programming

Lecture 6

Classes and Methods – Part II

Classes and methods – review…

- Classes have
  - Instance variables to store data
  - Method definitions to perform actions
- Instance variables should be private
- Classes need accessor, mutator methods
- Methods may be
  - Value returning methods
  - Void methods that do not return a value
- Let's see the Student example

Course content

- Previously…
  - Class and method definitions/examples
  - Methods
    - void/return
    - Local variables
    - Parameters
  - Information hiding (encapsulation)
    - The `public` and `private` Modifiers
  - UML Class Diagrams
- Now
  - Review – Student example
  - Constructors
  - Agency example
  - Static Variables & Methods
  - Overloading
  - Information Hiding Revisited

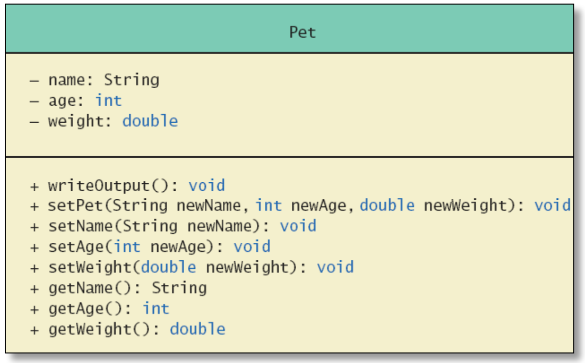
Constructors

- A special method called when instance of an object created with new
  - Create objects
  - Initialize values of instance variables
- Can have parameters
  - To specify initial values if desired
- May have multiple definitions
  - Each with different numbers or types of parameters

s1 = new Student();

Defining Constructors

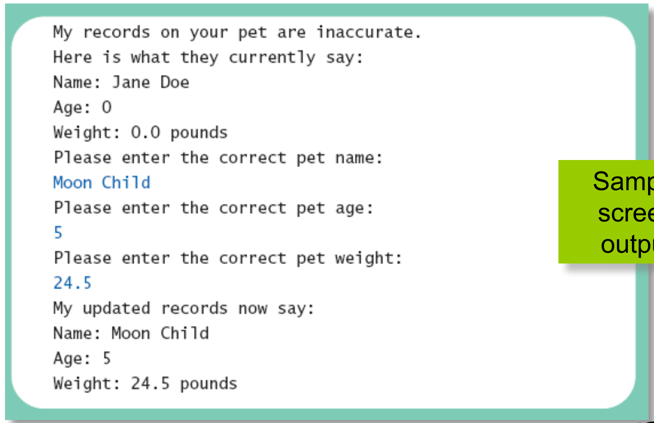
- Example class to represent pets



```
Pet
– name: String
– age: int
– weight: double
+ writeOutput(): void
+ setPet(String newName, int newAge, double newWeight): void
+ setName(String newName): void
+ setAge(int newAge): void
+ setWeight(double newWeight): void
+ getName(): String
+ getAge(): int
+ getWeight(): double
```

Defining Constructors

- Note CodeSamplesWeek7 – class `PetDemo`



```
My records on your pet are inaccurate.
Here is what they currently say:
Name: Jane Doe
Age: 0
Weight: 0.0 pounds
Please enter the correct pet name:
Moon Child
Please enter the correct pet age:
5
Please enter the correct pet weight:
24.5
My updated records now say:
Name: Moon Child
Age: 5
Weight: 24.5 pounds
```

Sample screen output

Defining Constructors

- Note CodeSamplesWeek7 – class `Pet`

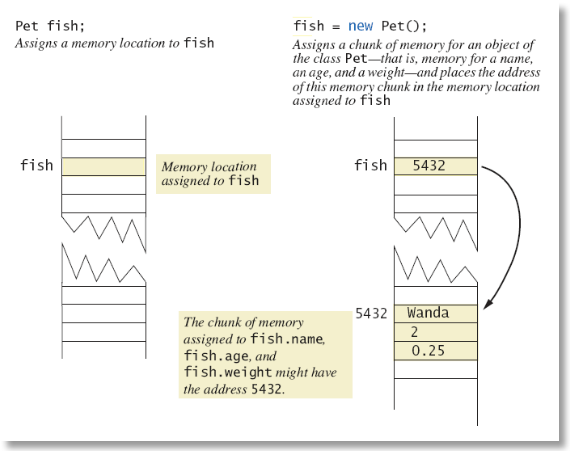
- Note different constructors
  - Default
  - With 3 parameters
  - With String parameter
  - With double parameter

Defining Constructors

- Constructor without parameters is the default constructor
  - Java will define this automatically if the class designer does not define any constructors
  - If you do define a constructor, Java will not automatically define a default constructor
- Usually default constructors not included in class diagram

Defining Constructors

- Figure 6.2 A constructor returning a reference

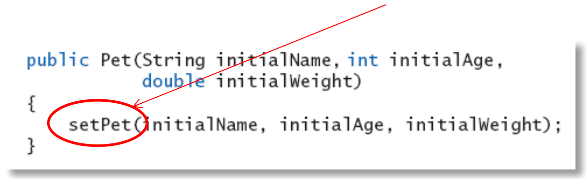

Pet fish;
Assigns a memory location to fish
fish = new Pet();
Assigns a chunk of memory for an object of the class Pet—that is, memory for a name, an age, and a weight—and places the address of this memory chunk in the memory location assigned to fish
fish
Memory location assigned to fish
fish
5432
The chunk of memory assigned to fish.name, fish.age, and fish.weight might have the address 5432.
5432
Wanda
2
0.25

Calling Constructor from Other Constructors

- **Pet2** class has the initial constructor and method set
- In the other constructors we can use the **this** reference to call initial constructor
- View CodeSamplesWeek7 – class **Pet3**
  - Note calls to initial constructor

Calling Methods from Other Constructors

- Constructor can call other class methods



```
public Pet(String initialName, int initialAge,
           double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

- Note CodeSamplesWeek7 – class **Pet2**
  - Note method **set**
  - Keeps from repeating code

Example - Agency

- There are 3 classes.
  - Person class - represents one person
  - Couple class - represents 2 persons
  - AgencyInterface class – interface with the user
    - The main method will be in Agency interface.
- First - let's see the code
- Next – see what happen when we execute the code

Person class

```
public class Person
{
    private String name;
    private int age;

   public Person()
    {
        name = "";
        age = 0;
    }
   public void setName(String newName)
    {
        name = newName;
    }
   public String getName()
    {
       return name;
    }
   public void setAge(int newAge)
    {
        age = newAge;
    }
   public int getAge()
    {
        return age;
    }
}
```

Class name:
Person

Instance variables

Constructor method:
Person

methods:
setName
getName
setAge
setAge

Couple class

```
public class Couple
{
    private Person he,she;

   public Couple()
    {
        he  = new Person();
        she = new Person();
    }
   public void setData(int option, String name, int age)
    {
        if (option==1) setData1(she,name,age);
        else           setData1(he,name,age);
    }
    private void setData1(Person p, String name, int age)
    {
            p.setName(name);
            p.setAge(age);
    }
   public String test()
    {
        if (she.getAge() < he.getAge()) return("GOOD FOR "+he.getName()+"!");
        else                            return("GOOD FOR "+she.getName()+"!");
    }
}
```

Class name:
Couple

Instance variables

Constructor method:
Couple

More methods:
setData
setData1
test

AgencyInterface class

```
import java.util.*;

public class AgencyInterface
{
    public static void main (String[] args)
    {
       Scanner console = new Scanner(System.in);
       Couple c = new Couple();
       int       herAge,hisAge,end;
       String    herName,hisName;
```

Class name:
AgencyInterface

variables:
Object **c** from Couple class

AgencyInterface class

```
do {
           System.out.print("her name: ");  herName = console.next();
           System.out.print("her age: ");   herAge=console.nextInt();
           System.out.print("his name: ");  hisName = console.next();
           System.out.print("his age: ");   hisAge= console.nextInt();

           c.addData(1,herName,herAge);
           c.addData(2,hisName,hisAge);

           System.out.println("********************");
           System.out.println(c.test());
           System.out.println("********************");

           System.out.print("Quit? (0)yes (1)no: ");
           end = console.nextInt());
           }
       while (end!=0);
       }
}
```

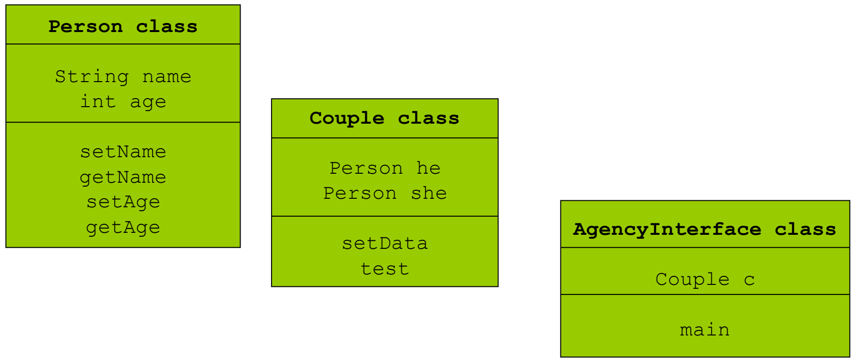
Example Agency

- Notice that AgencyInterface uses Couple and
- Couple uses Person.

- What happen when we run this program?

Example Agency

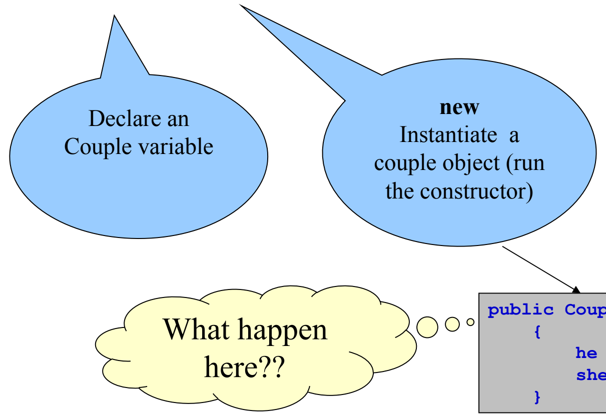
- What we have…



Start…AgencyInterface

- Go to main method

```
Couple c = new Couple();
```


Declare an Couple variable
new Instantiate a couple object (run the constructor)
What happen here??

```
public Couple()
    {
        he  = new Person();
        she = new Person();
    }
```

Couple...and…person

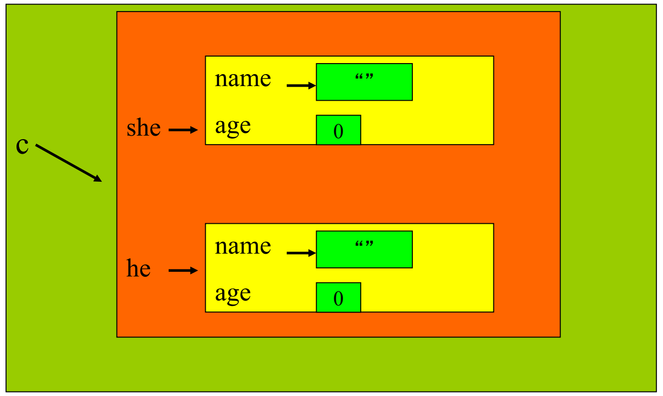
- Instantiate couple object (c) and run the constructor inside Couple.java

```
public Couple()
    {
        he  = new Person();
        she = new Person();
    }
```

- Instantiate two person objects (he and she) and run the constructor inside Person.java

```
public Person()
{
        name = "";
        age = 0;
}
```

Agency



What do we have until now?

- Remember that
  - c, she, he are references to objects!

What is this?!

- To access variables from c
  - c.he or c.she
  - It will not work since the variables are private
- To access methods from c
  - c.setData, c.test
  - It will work since the methods are public

What do we have until now?

- To access variables from he or she
  - he.age, she.age, he.name, she.name
  - It will **not** work since the variables are private
- To access methods from he and she
  - he.setName, she.setName, etc.
  - It will work since the methods are public

Let’s continue running…

- After create the objects the program stops.
- It will continue when you enter with the input (names and ages).

```
System.out.print("her name: ");
herName = console.next();
System.out.print("her age: ");
herAge = console.nextInt();
System.out.print("his name: ");
hisName = console.next();
System.out.print("his age: ");
hisAge= console.nextInt();

c.addData(1,herName,herAge);
c.addData(2,hisName,hisAge);
```

Let’s see what will happen in these lines

```
c.setData(1, herName, herAge);
```

AgencyInterface

```
public void setData(int option, String name, int age)
    {
      if (option==1)  setData1(she,name,age);
      else            setData1(he,name,age);
    }

private void setData1(Person p, String name, int age)
    {
          p.setName(name);
          p.setAge(age);
    }
```

Couple

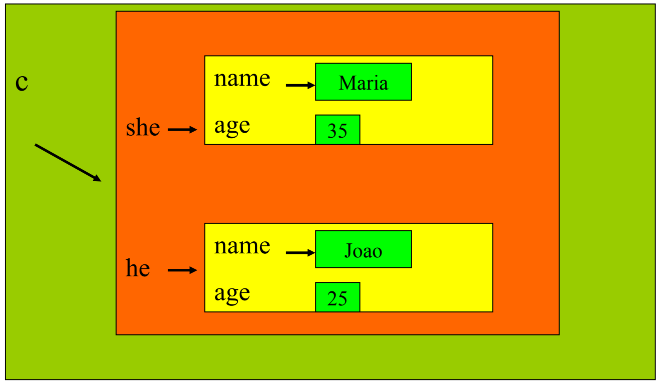
Person

```
public void setName(String newName)
{
        name = newName;
}
```

```
public void setAge(int newAge)
{
        age = newAge;
}
```

What do we have until now?

- Suppose the user put some data. So, you have:



- Next

Let's see what will happen in this line

```
System.out.println("*******************");
System.out.println(c.test());
System.out.println("*******************");
```

```
System.out.print(c.test());
```

```
public String test()
    {
        if (she.getAge() < he.getAge()) return("GOOD FOR " +he.getName()+"!");
        else                            return("GOOD FOR "+she.getName()+"!");
    }
```

```
  public String getName()
           {
        return name;
           }
          public void setAge(int newAge)
           {
        age = newAge;
           }
          public int getAge()
           {
        return age;
           }
```

Static Variables

- Static variables are shared by all objects of a class
  - Variables declared **`static final`** are considered constants – value cannot be changed
- Variables declared **`static`** (without **`final`**) can be changed
  - Only one instance of the variable exists
  - It can be accessed by all instances of the class

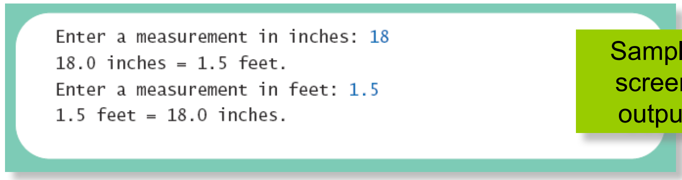
Static Methods

- Some methods may have no relation to any type of object
- Example
  - Compute max of two integers
  - Convert character from upper- to lower case
- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name

Static Variables

- Static variables also called *class variables*
  - Contrast with *instance variables*
- Do not confuse class variables with variables of a class type
- Both static variables and instance variables are sometimes called *fields* or *data members*

Static variables and methods - example



```
Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
```

Sample screen output

Static variables and methods - example

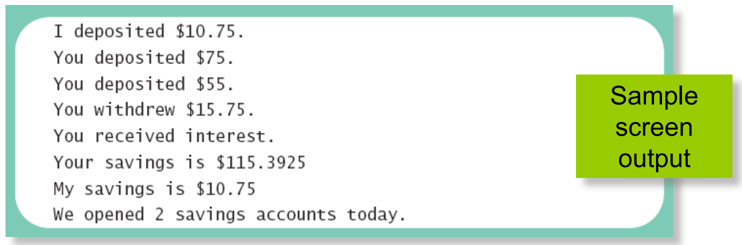
```
/**Class of static methods to perform dimension conversions.*/

public class DimensionConverter
{
   public static final int INCHES_PER_FOOT = 12;
   public static double convertFeetToInches (double feet)
   {
        return feet * INCHES_PER_FOOT;
   }
   public static double convertInchesToFeet (double inches)
   {
        return inches / INCHES_PER_FOOT;
   }
}
```

Static variables and methods - example

```
import java.util.Scanner;
/**Demonstration of using the class DimensionConverter.*/
public class DimensionConverterDemo
{
   public static void main (String [] args)
   {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter a measurement in inches: ");
        double inches = keyboard.nextDouble ();
        double feet = DimensionConverter.convertInchesToFeet(inches);
        System.out.println (inches + " inches = " + feet + " feet.");
        System.out.print ("Enter a measurement in feet: ");
        feet = keyboard.nextDouble ();
        inches = DimensionConverter.convertFeetToInches(feet);
        System.out.println (feet + " feet = " + inches + " inches.");
   }
}
```

Static and Nonstatic Methods - example

- View CodeSamplesWeek7 – classes **`SavingAccount`** and **`SavingAccountDemo`**



```
I deposited $10.75.
You deposited $75.
You deposited $55.
You withdrew $15.75.
You received interest.
Your savings is $115.3925
My savings is $10.75
We opened 2 savings accounts today.
```

Tasks of `main` in Subtasks

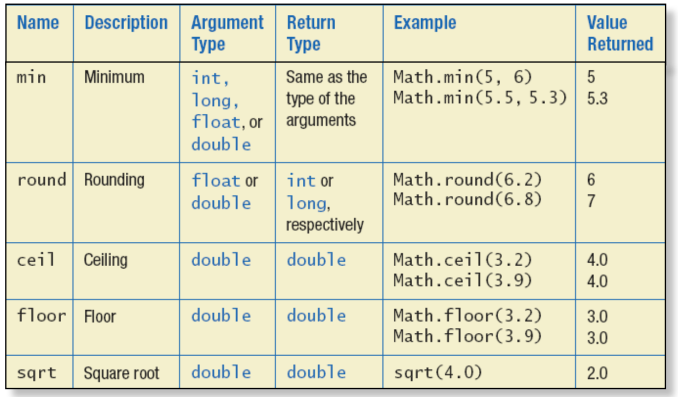
- Program may have
  - Complicated logic
  - Repetitive code
- Create static methods to accomplish subtasks
- Consider CodeSamplesWeek7 – classes **`SpeciesEqualDemo1`** and **`SpeciesEqualDemo2`**
  - **`SpeciesEqualDemo1`**
    - a **`main`** method with repetitive code
  - **`SpeciesEqualDemo2`**
    - uses helping methods

Adding Method `main` to a Class

- Method main used so far in its own class within a separate file
- Often useful to include method main within class definition
  - To create objects in other classes
  - To be run as a program
- Note CodeSamplesWeek7 a redefined class `Species`
- See the details later

The `Math` Class

- Example methods, figure 6.3b



Name	Description	Argument Type	Return Type	Example	Value Returned

`min`	Minimum	`int`, `long`, `float`, or `double`	Same as the type of the arguments	`Math.min(5, 6)`
				`Math.min(5.5, 5.3)`
					5
					5.3

`round`	Rounding	`float` or `double`	`int` or `long`, respectively	`Math.round(6.2)`
				`Math.round(6.8)`
					6
					7

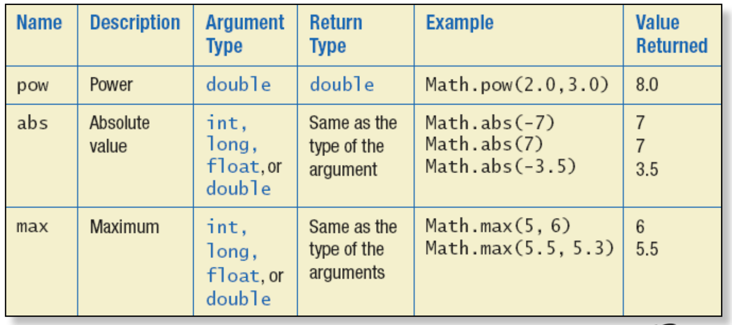
`ceil`	Ceiling	`double`	`double`	`Math.ceil(3.2)`
				`Math.ceil(3.9)`
					4.0
					4.0

`floor`	Floor	`double`	`double`	`Math.floor(3.2)`
				`Math.floor(3.9)`
					3.0
					3.0

`sqrt`	Square root	`double`	`double`	`sqrt(4.0)`	2.0

The `Math` Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed
- Example methods, figure 6.3a



Name	Description	Argument Type	Return Type	Example	Value Returned
`pow`	Power	`double`	`double`	`Math.pow(2.0,3.0)`	8.0

`abs`	Absolute value	`int`, `long`, `float`, or `double`	Same as the type of the argument	`Math.abs(-7)`
				`Math.abs(7)`
				`Math.abs(-3.5)`
					7
					7
					3.5

`max`	Maximum	`int`, `long`, `float`, or `double`	Same as the type of the arguments	`Math.max(5, 6)`
				`Math.max(5.5, 5.3)`
					6
					5.5

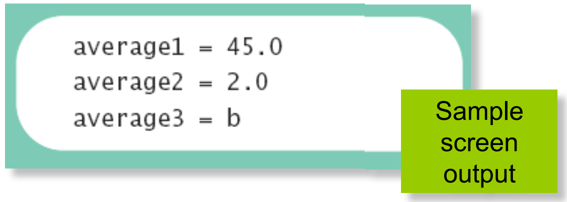
Random Numbers

- `Math.random()` returns a random double that is greater than or equal to zero and less than 1
- Java also has a `Random` class to generate random numbers
- Can scale using addition and multiplication; the following simulates rolling a six sided die

```
int die = (int) (6.0 * Math.random()) + 1;
```

Overloading Basics

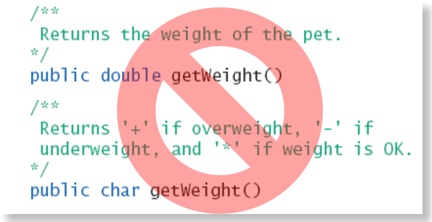
- View CodeSamplesWeek7 - **class Overload**
- Note overloaded method **getAverage**



```
average1 = 45.0
average2 = 2.0
average3 = b
```

Overloading and Return Type

- You must not overload a method where the only difference is the type of value returned



```
/**
 Returns the weight of the pet.
*/
public double getWeight()
/**
 Returns '+' if overweight, '-' if
 underweight, and '*' if weight is OK.
*/
public char getWeight()
```

Overloading and Type Conversion

- Overloading and automatic type conversion can conflict
- Recall definition of Pet class of CodeSamplesWeek7
  If we pass an integer to the constructor we get the constructor for age, even if we intended the constructor for weight
- Remember the compiler attempts to overload before it does type conversion
- Use descriptive method names, avoid overloading

Your task

- Read
  - Lecture slides
  - Chapter 6 of the text book
- Exercises
  - MyProgrammingLab
  - Implement the examples in CodeSamplesWeek7 (available in Blackboard)
  - Use debug in BlueJ to understand what is happening

Next week – midterm exam – in class

- Introduction and Java basics
- Conditional structures
  - Example from past exam - triangle
- Loop structures
  - Example from past exam - population
- Classes and methods
  - Example from past exam - student

Midterm exam

- You will receive the templates:

```
Using TIO:
public class NameOfYourClass{
  public static void main(String[] args)   {
    Scanner console = new Scanner(System.in);
    // your code
    // to read you can use:  console.nextInt(), console.next(); etc.
    // to print you can use: System.out.print();
  }
}

Using GUI:
public class NameOfYourClass {
  public static void main (String[] args) {
    // to read use: JOptionPane.showInputDialog("message")
// to write use:
// OptionPane.showMessageDialog(null,str,"message",JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Midterm exam – WED – 12/04 – 9:00-11:00

- It will have
  - 5 multiple choice questions and
  - 3 programming questions
- involving
  - Java basics, Input/Output
  - Control structures
    - Conditional statements
    - Loop statements
  - Methods and Classes

☺
Good luck!!