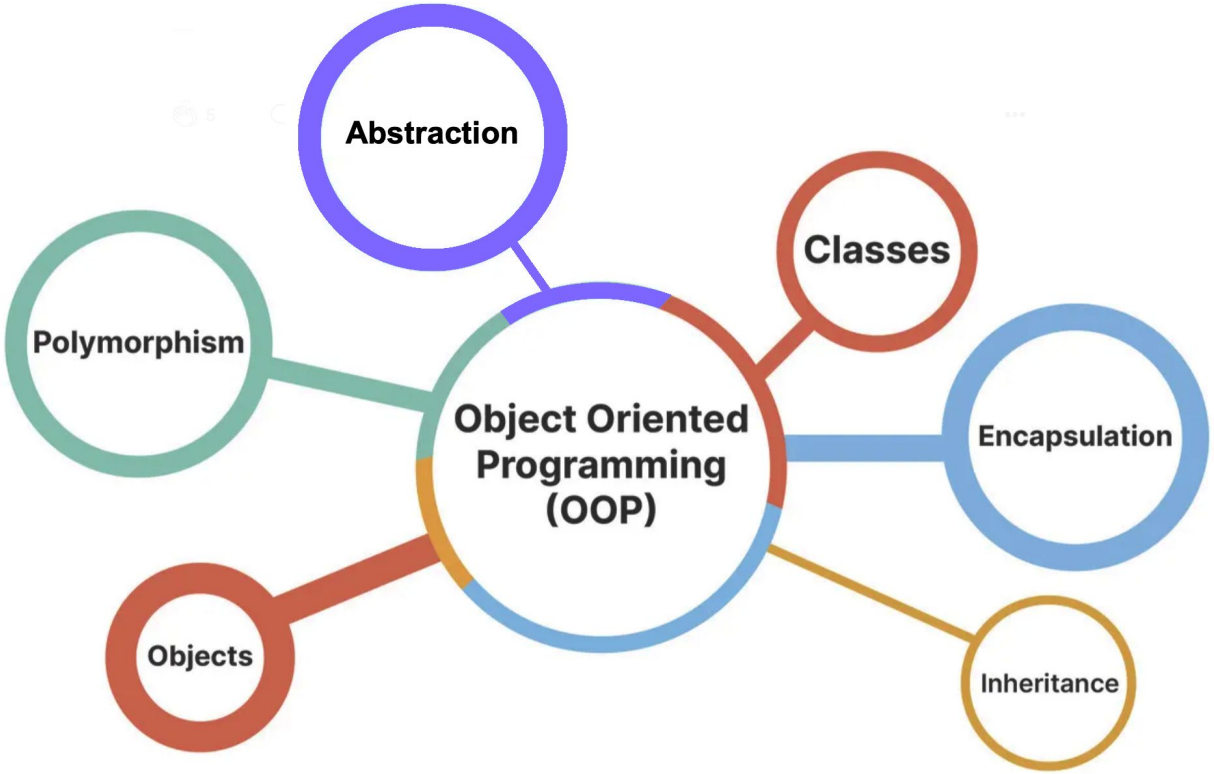
COSC 121: Computer Programming II



Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?

Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?
  - Classes and objects what will be involved

Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?
  - Classes and objects — what will be involved
  - Identifying attributes — what objects store about themselves

Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?
  - Classes and objects — what will be involved
  - Identifying attributes — what objects store about themselves
  - Class responsibilities — who interacts with whom

Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?
  - Classes and objects — what will be involved
  - Identifying attributes — what objects store about themselves
  - Class responsibilities — who interacts with whom
  - Abstraction — exposing only essential info

Recap: Object-Oriented Design

- Process of building software based on a series of objects that interact together to solve a problem
- Object-oriented programming (OOP)
  - Set of programming techniques to support this design
- OOP examples from COSC 111/prereq?
  - Classes and objects — what will be involved
  - Identifying attributes — what objects store about themselves
  - Class responsibilities — who interacts with whom
  - Abstraction — exposing only essential info
  - Encapsulation — restricting data access

iClicker Question

What is the purpose of a class definition?

A. To increase code reusability and save effort when multiple objects are created from it
B. To organize complex structures in real-world programs
C. To define a blueprint/template for all its objects
D. All of the above

Example:

```
public class BankAccount {
    private String ownerName;
    private double balance;

    public double getBalance() {
        return balance;
    }
    public void withdraw( double amount ) {
        if( balance > amount )
            balance -= amount;
    }
    public void deposit( double amount ) {
        if( amount > 0 )
            balance += amount;
    }
}
```

iClicker Question

What is an object in Java?

A. Everything in Java is an object
B. An object is an instance of a class with its own state and behavior
C. An object is the variable that refers to the object when the object is created from a class
D. An object is just another name for a class in Java

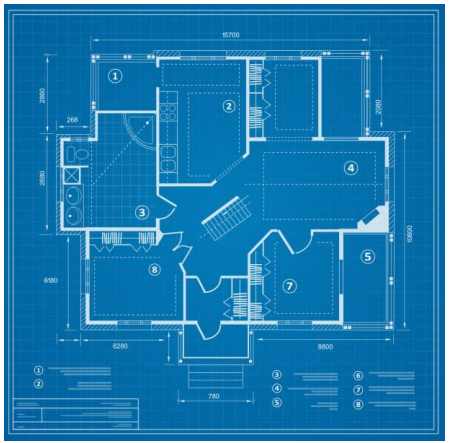
What objects do you see in this game?

Recall: objects have its own memory (state) and behavior

How Are Objects Created?

Step 1: Blueprint (Design)

- Attributes
- Behavior



Step 2: Construction

- Instantiate an object from the class

In Java, all objects of a design have the same attributes and behaviors

Step 1: Designing Objects

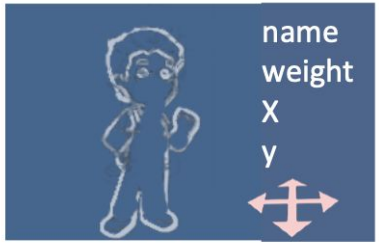
- A class represents the blueprint of a group of objects with the same design
- The class defines the attributes and behaviors for objects
- Attributes:
  - Defined as instance variables inside the class
- Behaviors:
  - Defined as methods inside the class

Example attributes?
Example behaviors?

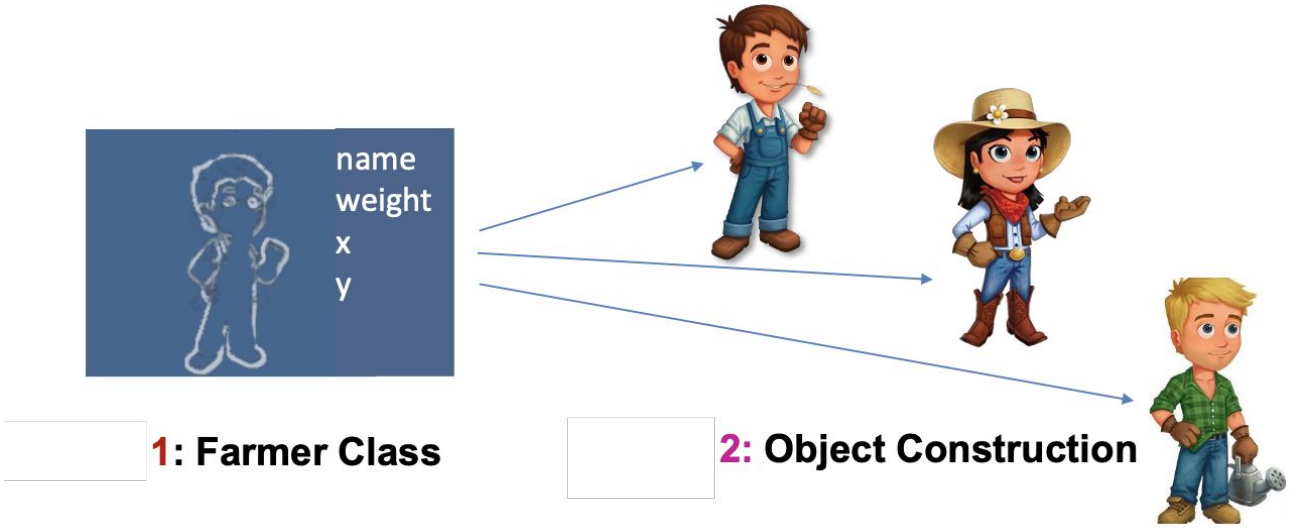
Example Farmer Class

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;

      //methods (actions)
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b){
         x = a;     y = b;
      }
}
```

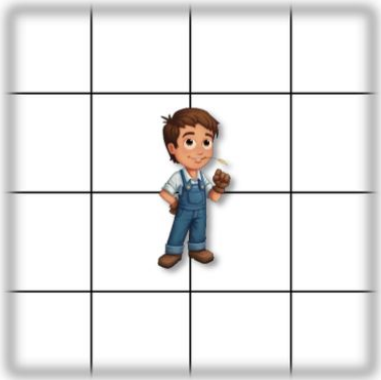


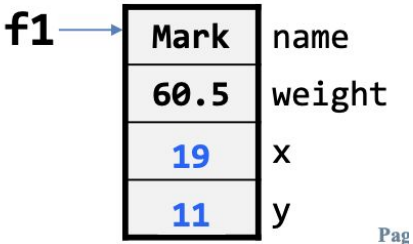
Step 2: Object Instantiation



Instantiating Objects in main()

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      f1.name = "Mark";
      f1.weight = 60.5;
      f1.x = 20;
      f1.y = 10;
      f1.moveRight();
      f1.moveDown();
      f1.moveTo(19,11);
  }
}
```

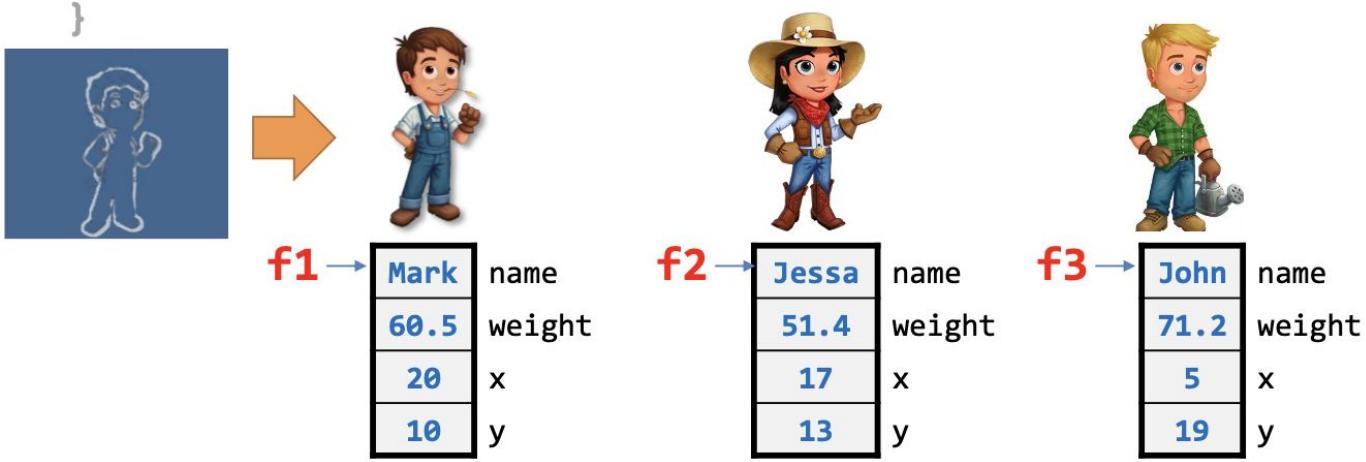




Creating Several Objects

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer();
      Farmer f2 = new Farmer();
      Farmer f3 = new Farmer();
      ... //change attribute values for f1,f2,f3
  }
}
```

Each object has its own memory



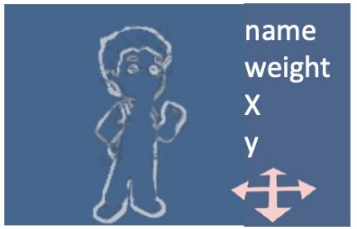
Constructors

- Used to instantiate objects from a class definition

Where is the constructor?

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;

      //methods (actions)
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b){
         x = a;     y = b;
      }
}
```



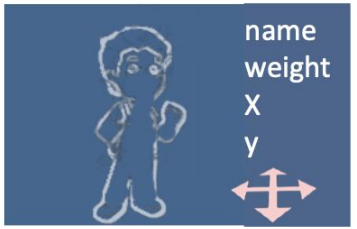
Constructors

- Used to instantiate objects from a class definition

Where is the constructor?
All Java classes have a default constructor!

```
class Farmer {
      //instance variables (attributes)
      String name;
      double weight;
      int x, y;

      //methods (actions)
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b){
         x = a;     y = b;
      }
}
```



Constructors

- Used to instantiate objects from a class definition
- Special characteristics:
  - Must have the same name as its class
  - Do not have a return type (not even void)
  - Invoked using the new operator when object is created
- Default constructors have no input parameters

```
Farmer()
{
}
```

Sets all attributes to their default values:
- String/other references to null
- Numeric to zero
- Boolean to false
- Char to \u0000

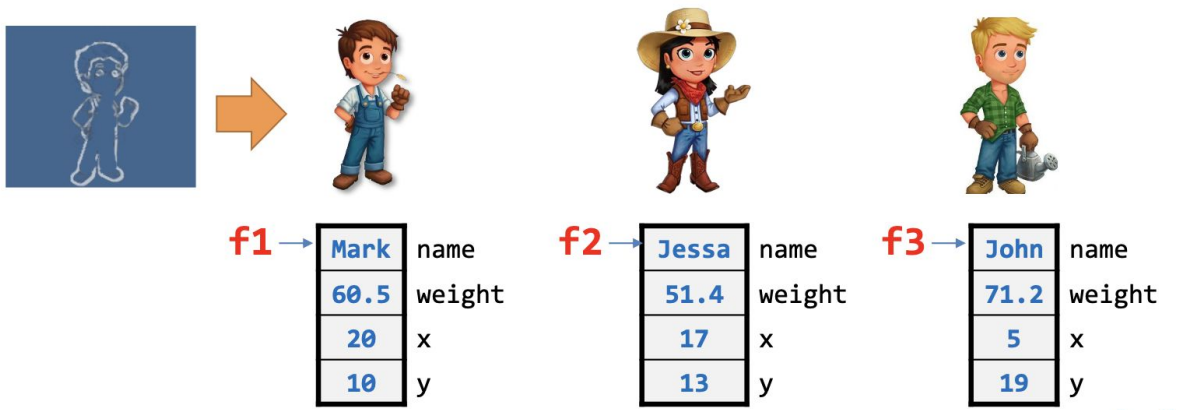
- What if you want to create an object with specific attribute values?

Defining Your Own Constructor

```
class Farmer {
      //instance variables
      String name;
      double weight;
      int x, y;
      //constructors
      Farmer(String aName, double aWeight, int x1,int y1){
         name = aName;
         weight = aWeight;
         x = x1;
         y = y1;
      }
      //methods
      public void moveUp()    {y++;}
      public void moveDown()  {y--;}
      public void moveRight(){x++;}
      public void moveLeft()  {x--;}
      public void moveTo(int a, int b) { x = a; y = b; }
}
```

Updated main()

```
public class FarmerTest {
  public static void main(String[] args) {
      Farmer f1 = new Farmer("Mark", 60.5, 20, 10);
      Farmer f2 = new Farmer("Jessa", 51.4, 17, 13);
      Farmer f3 = new Farmer("John", 71.2, 5, 19);
  }
}
```



iClicker Question

What are the attributes in this class?

A. BankAccount
B. getBalance(), withdraw(), deposit()
C. ownerName, balance
D. BankAccount, ownerName, balance

Example:

```
public class BankAccount {
    private String ownerName;
    private double balance;

    public double getBalance() {
        return balance;
    }
    public void withdraw( double amount ) {
        if( balance > amount )
            balance -= amount;
    }
    public void deposit( double amount ) {
        if( amount > 0 )
            balance += amount;
    }
}
```

iClicker Question

Which method has the responsibility of adding money to an account?

A. BankAccount
B. getBalance()
C. withdraw()
D. deposit()

Example:

```
public class BankAccount {
    private String ownerName;
    private double balance;

    public double getBalance() {
        return balance;
    }
    public void withdraw( double amount ) {
        if( balance > amount )
            balance -= amount;
    }
    public void deposit( double amount ) {
        if( amount > 0 )
            balance += amount;
    }
}
```

iClicker Question

What mechanism enables encapsulation in Java?

Recall that encapsulation is about data access to the outside world.

A. private and public modifiers
B. private getter and setter methods
C. getter and setter methods
D. the static modifier

Additional Notes

- Visibility modifiers (public, private, protected)
  - getters and setters can be used for private attributes
- static modifier
  - Static **variables** are shared by all instances of the class
    Ex: public static int numberOfCarsCreated = 0;
    System.out.println(Car.numberOfCarsCreated);
  - Static **methods** are tied to the class (e.g., utility methods)
    Ex: Math.max(3, 5);
- this keyword
  - Used to refer to the current object inside its method
  - Avoids name conflicts

Programming Practice (~ 10 min)

Notation:
use private for -
and public for +

- Use Eclipse to create two classes in Java

Circle
-radius: double -color: String -filled: Boolean
+Circle() +Circle(radius: double) +Circle(radius: double, color: String, filled: boolean) +getters/setters for all attributes +getArea(): double +getPerimeter(): double +toString(): void

Rectangle
-width: double -height: double -color: String -filled: Boolean
+Rectangle() +Rectangle(width: double, height: double) +Rectangle(width: double, height: double, color: String, filled: boolean) +getters/setters for all attributes +getArea(): double +getPerimeter(): double +toString(): void

Solution

```
public class Circle {
    // attributes
    private String color;
    private boolean filled;
    private double radius;
    // constructors
    public Circle() { this(1,"Black",true); }
    public Circle(double radius) { this(radius, "Black", true);}
    public Circle(double radius, String color, boolean filled) {
        setRadius(radius);
        setColor(color);
        setFilled(filled);
    }
    // methods
    public double getArea() {return Math.PI*radius*radius;}
    public double getPerimeter(){return 2*Math.PI*radius;}
    // setters/getters
    public String getColor()              { return color;}
    public void setColor(String color)    { this.color=color;}
    public boolean isFilled()             { return filled;}
    public void setFilled(boolean filled){ this.filled=filled;}

    public double getRadius()             { return this.radius;}
    public void setRadius(double radius){
        if(radius >= 0) this.radius = radius;
    }
    // to string
    public String toString() {
        return "radius="+radius+",color="+color+",filled="+filled;
    }
}
```

```
public class Rectangle {
    // attributes
    private String color;
    private boolean filled;
    private double width,height;
    // constructors
    public Rectangle() { this(1,1,"Black",true);}
    public Rectangle(double width,double height) { this(width, height,"Black",true); }
    public Rectangle(double width, double height, String color, boolean filled) {
        setWidth(width);   setHeight(height);
        setColor(color);
        setFilled(filled);
    }
    // methods
    public double getArea() {return width * height;}
    public double getPerimeter() {return 2 * (width + height);}

    // setters/getters
    public String getColor()              {return color;}
    public void setColor(String color)    { this.color = color;}
    public boolean isFilled()             {return filled;}
    public void setFilled(boolean filled) { this.filled = filled;}
    public double getWidth()              {return width;}
    public void setWidth(double width)    { if(width >= 0) this.width = width;}
    public double getHeight()             {return height;}
    public void setHeight(double height) {if(height >= 0) this.height = height;}

    // to string
    public String toString() {
        return "color="+color+", filled="+filled+", width="+width+", height="+height;
    }
}
```

Note how much code redundancy we have! ***Inheritance*** can solve this!

With Inheritance

Shape

-color: String
-filled: Boolean

+Shape()
+Shape(color: String, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+toString(): String

Parent class with generic/common vars and methods

Child class inherits from Shape, defined with specific info

Another child class

Circle

-radius: double

+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void

Rectangle

-width: double
-height: double

+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double, color: String, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double

Object References

- Recall that primitive variables uses pass-by-value while objects use pass-by-reference

What does this print after calling foo()?

```
public static void main(String[] args) {
  int x = 0;
  Circle c = new Circle(0);

  System.out.printf("Before foo: x is %d, c.radius is %.0f\n",x,c.getRadius());
  foo(x, c);
  System.out.printf("After foo: x is %d, c.radius is %.0f\n",x,c.getRadius());
}

public static void foo(int a, Circle b) {
  a = 7;
  b.setRadius(7);
}
```

Output:
Before foo: x is 0, c.radius is 0
…

Object References

- Recall that primitive variables uses pass-by-value while objects use pass-by-reference

What does this print after calling foo()?

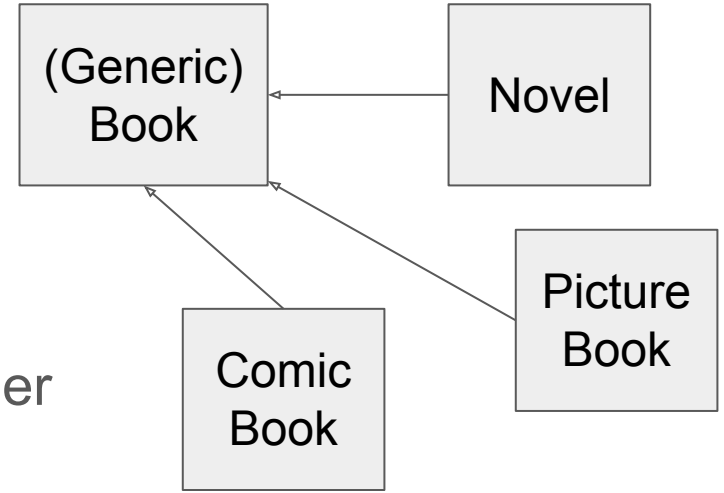
```
public static void main(String[] args) {
  int x = 0;
  Circle c = new Circle(0);

  System.out.printf("Before foo: x is %d, c.radius is %.0f\n",x,c.getRadius());
  foo(x, c);
  System.out.printf("After foo: x is %d, c.radius is %.0f\n",x,c.getRadius());
}

public static void foo(int a, Circle b) {
  a = 7;
  b.setRadius(7);
}
```

Output:
Before foo: x is 0, c.radius is 0
After foo: x is 0, c.radius is 7

Next Class: Inheritance



- Another OOP technique
- Purpose:
  - Organize "related" classes together
  - Maximize reusable classes
- What is reusability and its advantages?
  - Defined class once, don't define it again
  - Defined methods once, don't define them again
  - Changes isolated to one place
  - Bugs isolated to one place