1 **Processing:**
Object-Oriented Programming
CST112

2 **Classes and Objects (Page 1)**

- Classes are programmed representations of entities (people and things) in the real world
- In an object-oriented language, you use classes and objects to organize and manipulate your *data*

3 **Classes and Objects (Page 2)**

- Imagine writing a computer program to keep track of the houses in a new condominium development (still under construction).
- The houses differ only slightly from one another
  - Each house has a distinctive siding color, an indoor paint color, a kitchen cabinet style, etc.
- In an object-oriented computer program, each house is an object

4 **Classes and Objects (Page 3)**

- Houses differ slightly from one another, but all the houses share the same "list" of characteristics, i.e.
  - Siding color
  - Interior paint color
  - Kitchen cabinet style
  - *Etc.*
- An object-oriented program needs a *master list* (called the class) containing all the characteristics that a house object can possess

5 **Classes and Objects (Page 4)**

- An architect's *blueprint* is like an object-oriented programmer's class—it is a list of characteristics that each house will have, i.e.
  - The blueprint says "siding"—the actual house object has gray siding
  - The blueprint says "kitchen cabinet"—the actual house object has Louis XIV kitchen cabinets

6 **Classes and Objects (Page 5)**

- After the blueprint it created, it can be used to build several houses
- The same is true with classes and objects:
  - First the programmer writes code to describe a class
  - When the program runs, the computer creates objects (one or more) from the (blueprint) class

7 **Classes and Objects (Page 6)**

- For the "StickMan" class the "list" of attributes are:
  - Current x- and y-coordinates (increments by speed)
  - Speed
- For the "Ball" class the "list" of attributes are :
  - Current x- and y-coordinates (increments by horizontal and vertical speeds)
  - Horizontal (x) and vertical (y) *speed* (both change from positive to negative, and

back)
- Red, green and blue values

8 **Classes (Page 1)**

- A class is a *programmer-defined* type containing methods (in classes functions are called methods) that manipulate data
- One or more objects are instantiated from the classes and work together to build an application
- OOP (object-oriented programming) encapsulates (*each object has its own*):
  - Attributes (the data/properties)
  - Behaviors (the functions/methods)

9 **Classes (Page 2)**

- The layout of each class includes:
  - The outer wrapping made up of a class header (also called a signature) names the class
  - The inner body of the class in enclosed in left and right curly braces {always used in pairs} and provides the class's functionality
- By *convention* each word in a class name should begin with an uppercase letter, i.e.
  - StickMan

10 **Classes (Page 3)**

- Format:

```
class ClassName
{
    data
    constructors
    methods
}
```

  - The *data* is the class's list of "attributes" (properties) also known as instance variables (or fields)
  - The *constructor* initializes each object
  - The *methods* provides the object's "behaviors"

11 **Classes (Page 4)**

- Example:

```
class StickMan
{
  float x;
  float y;

  StickMan()
  {
    x = 0;
    y = height / 2;
  }
   …
}
```

12 **Classes (Page 5)**

- Example (alternative brace placement):

```
class StickMan {
  float x;
  float y;

  StickMan() {
    x = 0;
    y = height / 2;
  }
  …
}
```

13 **Constructors (Page 1)**

- A special function (method) which creates the instance of the class (an object) and *initializes* the instance variables within the object
- The constructor has the *same name* as the class
- Executed whenever an application *instantiates* an object from the class because the constructor method is being *called*
- Guarantees that instance variables (the class' data) always will be in a *consistent state*

14 **Constructors (Page 2)**

- Can take arguments but do *not* return values—never specify a type, not even void
- Format:

ConstructorName([*type parameter1*, *type parameter2*, …])
// Name is the same as the class name

- Example:

```
StickMan()
{
   x = 0;
   y = height / 2;
}
```

15 **Constructors (Page 3)**

- The constructor usually follows the class header and the instance variables (fields):

```
class StickMan
{
  float x;
  float y;

  StickMan()
  {
     x = 0;
     y = height / 2;
  }
```

20 **Using Objects**

- There are three steps is using classes and objects:
  1. Declare object variables—this is the same as for any variable and usually takes place globally before the setup() function
  2. Instantiate and initialize the object—usually inside the setup() function by calling its constructor method
  3. Call the methods of the object—usually inside the draw() function so that they are called repeatedly

21 **Declaring an Object Variable (Page 1)**

- Multiple instances of the class called objects can be declared and used in an application
- Within a Processing application, object variables usually are global within the class, so they are declared first (before the setup() function)

22 **Declaring an Object Variable (Page 2)**

- The class name is used like the *data type* in the declaration of an object variable
  - Classes actually are types just like primitives, e.g. int and float
- Format:
  ClassName objectName;
- Example:
  StickMan stick1;
  - Class names always start with an *uppercase* letter; object names start with a *lowercase* letter

23 **Instantiating an Object (Page 1)**

- An object is instantiated (created) within a class definition by calling the its constructor method
  - The return value of the new object is assigned (=) to the object variable
- Within a Processing application objects usually are instantiated within the setup() function

24 **Instantiating an Object (Page 2)**

- The keyword new instantiates (creates) the object
  - An object variable not yet been instantiated is considered to be a null object
- Format:
  objectName = new *ConstructorName*();
  - ConstructorName() is the same as the class name
- Example:
  stick1 = new StickMan();

26 **Declaring and Instantiating an Object at the Same Time**

- The syntaxes to *declare* and *instantiate* an object can be combined into a single statement
- Format:
  ClassName objectName = new *ConstructorName*();
  - The *ClassName* and *ConstructorName* are the same
- Example:
  StickMan stick1 = new StickMan();

27 **Calling an Object's Methods (Page 1)**

- In object-oriented programming, the functions inside a class definition are called methods
- Within a Processing application these methods usually are called from inside the draw() function

28 **Calling an Object's Methods (Page 2)**

- Class methods are called by joining the name of the object to the method name using a notation called "dot syntax"
- Format:
  objectName.methodName([*argment1*, *argument2*, …]);
  - As with every function call, *arguments* are optional
- Examples:
  stick1.display();
  stick1.move();

29 **Class Layout (Page 1)**

- In Processing a class is a new block of code
  - It can be positioned separately at the bottom of the other application elements, or …
  - It can be placed in a separate file within the application folder

30 **Class Layout (Page 2)**

- Suggested format:

```
// Declare objects here
void setup()
{
   // Instantiate objects here
}
void draw()
{
   // Call the class methods here
}
class ClassName
{
  //  Data, constructor, methods here
}
```

31 **Class Layout (Page 3)**

- Example:

```
StickMan stick1;
void setup()
{
   stick1 = new StickMan();
}
void draw()
{
   // Call the class methods here
}
```

```
class StickMan
{
  // Data, constructor, methods here
}
```

54 **Class and Driver Files**

- In Processing and in Java, it is common practice to place the class definition and the statements that use the class in separate files (the driver file)
- Both files are contained in the same project folder
- This enhances the "reusability" of the class
  - Easier to use the class in multiple applications
- In Processing the two files are accessed from two separate tabs above the editor window
  - Click on either tab to access the code for that file

55 **What Is the "Driver"**

- The driver is the file (in Java it is called the driver class) that "drives" the application
  - The term "driver" is used more in educational context, but not professionally
- It serves as the entry point where execution of the application begins (and ends)
- In Processing the driver is the file that declares and instantiates the objects, and contains functions setup() and draw() to call the class methods

63 **Adding Tabs to the Editor**

- To add a new tab to the Editor window so that the class file can be "separated" from the driver application and placed into the new window:
  - Click the Tab icon inside the square at the top right-hand corner of the window
  - Select New Tab from the "Tab" drop-down menu
  - Type a name for the tab (usually a good idea to name it after the class) and click the <OK> button
- A new ".pde" file with the same name as the tab is inserted into the project folder

81 **Constructors with Arguments (Page 1)**

- Like any other function, a constructor method can take parameters
- Constructor declaration format:
  ClassName objectVariable;
- Constructor call (class instantiation) format:
  objectVariable = new *ConstructorName*([*argumentList*]);
- Constructor class header format:
  ClassName([*parameterList*])

82 **Constructors with Arguments (Page 2)**

- Constructor declaration example:
  Ball ballOne;
- Constructor call (class instantiation) example:
  ballOne = new Ball(35);
- Constructor class header format:
  Ball(int s);