
# CSCI 1106
# Lecture 13

Classes, Objects, and Events

## Announcements

- Today's Topics
  - Objects
  - Classes
  - Instances
  - Events
  - Listeners

# A Real Life Example: A Pen

- Everyone understands what a pen is and what it does
- All pens share certain properties:
  - A pen has
    - an ink color
    - a type (ballpoint, fountain, roller-ball, fiber tip, gel ink, …)
    - it can click or have a cap
  - A pen does things (specifically it writes on paper)
  - A pen responds to events
    - If you put the tip to paper and move it, it writes
    - If you step on it, it will likely break
- A specific pen has a specific
  - Pen colour
  - Type
  - Cap (or not)
- A pen is an object!

# Objects

- An object is a programming abstraction
- Objects have:
  - A type (what kind of object is it?)
  - Properties
    - Specific attributes and characteristics
    - Often stored in variables in the object
  - Operations
    - Actions that can be performed on the objects
    - Actions that the objects can perform
    - Specified using methods or functions
- Objects are specified by classes

# Examples of Objects

**MovieClip**

- Properties
  - Position: *x, y*
  - Size: *width, height*
  - Orientation: *scaleX, scaleY, rotation*
  - …
- Operations
  - addEventListener()
  - removeEventListener()
  - …

**Stage**

- Properties
  - Position: *x, y*
  - Size: *stageHeight, stageWidth*
  - Mouse position: *mouseX, mouseY*
  - …
- Operations
  - addChild()
  - removeChild()
  - …

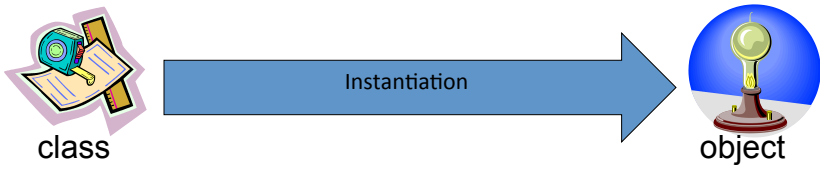
# Specifying Objects

- Question: How do we specify the properties and operations of an object?
- Idea: A class is a blueprint for an object
  - Lists all the properties (variables)
  - Contains a special function called a *constructor*
  - Contains the code for all operations (functions)
- **Note: A blueprint is NOT the object!**

```
class Pen {
  penColour : Color;
  penTip : PenType;
  panCap : boolean;
  penCapped : boolean;

  function Pen() {
    …
  }

  function write( text : string ) : void {
    …
  }

  function isPenCapped() : boolean {
    return penCapped;
  }

  function uncapPen() : void {
    …
  }

  function capPen() : void {
    …
  }
}
```

# Objects are Instances of Classes

- Idea: To build an object, you need its blueprint
  - E.g., cars, boats, plains, buildings, etc…
- Idea: An object is described by its class
  - E.g., We say "That is a car" or "That is a boat"
  - We can have many cars and many boats
  - But only one blueprint for each
- *Instantiation* is the action of creating an object from its class
- Idea: To instantiate an object, we call the class' constructor
  - E.g., `myPen = Pen();`




# Inheritance in a Nutshell

- Example:

  Halifax is a provincial capital, which is a city, which is a living place, which is a location, which is a noun.
- Observations:
  - This sentence moves from the specific to general
  - Each more specific object inherits the properties of its more general parent
- Corollary: Objects and classes are similar
  - E.g. a pen is a writing device is a device is a noun

# Inheritance in Action Script

## (All your base are belong to us)

- Idea: A class *extends* another class inherits from it
  - Inherits all properties
  - Inherits all functions
- Obs: Most of the objects in Flash are *MovieClip* objects
- Idea: Most classes describing objects in your game inherit from *MovieClip*
- Objects can also *listen for events*

```
class Pen extends MovieClip {
  // All properties and operations of
  // MovieClip belong to Pen

  function Pen() {
    …
  }

  function write( text : string ) : void {
    …
  }

  function isPenCapped() : boolean {
    return penCapped;
  }

  function uncapPen() : void {
    …
  }

  function capPen() : void {
    …
  }
}
```

# Examples from Tutorial 1

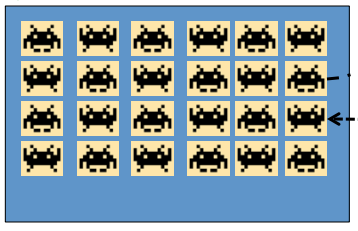
- Examples of classes
  - *Main* – class instantiated when game begins
  - *Level1* – class describing the background
  - *Paddle* – class describing the paddle
- Examples of objects
  - Instantiation of the background, e.g., in `Main.as`
    `new Level1();`
    instantiates a *Level1* object from the *Level1* class
  - `paddle` – an instantiation of *Paddle*
- Note: objects can be instantiated either
  - in code: as in `Main.as`
  - on the stage: by dragging objects on to the stage

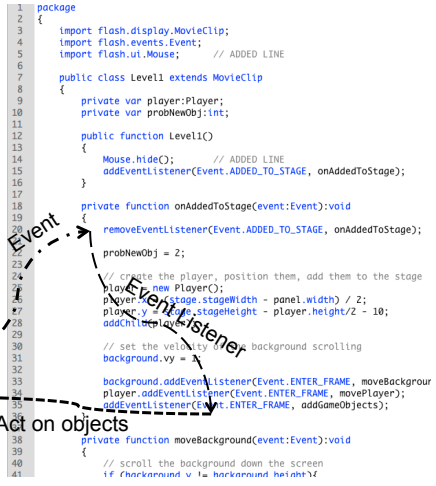
# Events

- Recall events are actions that a game responds to
  - Events occur either as a result of the user or the system
  - Events include:
    - Internal events: ENTER_FRAME, ADDED_TO_STAGE, REMOVED_FROM_STAGE
    - Mouse events : CLICK, DOUBLE_CLICK, RIGHT_CLICK, ROLL_OVER
    - Keyboard events : KEY_DOWN, KEY_UP
- Key Ideas:
  - Objects can listen for events
  - Flash looks after detecting when events occur and what they are
  - Flash does not know how to handle events
  - A *listener* is function that you implement to handle an event
  - Flash must be informed to call this *listener* when an event occurs

# Events in a Nutshell

- Loop:
  - Wait for event
  - Event occurs
  - Event listener called
  - Listener handles event
  - Listener returns



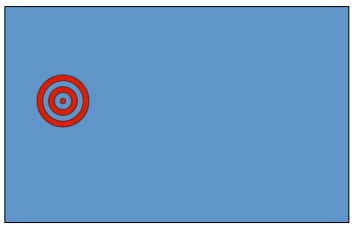


# Setting Event Listeners

- To set a listener on an object, use the `addEventListener(event, listener)` function where
  - *event*: is the event to listen for
  - *listener*: is the function to handle the event
- To remove a listener from an object, use function `removeEventListener(event, listener)`
- See example in Tutorial 2

# Setting Listeners

```
package
{
  import flash.display.MovieClip;
  import flash.events.Event;

  public class Main extends MovieClip
  {
    target : Target;                    // A MovieClip of a target

    public function Main()
    {
      target = Target();                // Instantiate a new target on the stage

      circle.addEventListener(Event.ENTER_FRAME,moveTarget);  // add listener
    }

    public function moveTarget(event:Event):void
    {
       target.x = mouseX;  // set target location
       target.y = mouseY;  // to mouse location
    }
}
```



# Events in a Nutshell

- Loop:
  - Wait for event
  - Event occurs
  - Event listener called
  - Listener handles event
  - Listener returns