# Windowing and Extending clases

# Extending (inheritance)

- Every class derives from Object
- Classes must extend from one, and only one, base class
- Some situations might “require” two base classes
- Java allows for unlimited derivitation from a type of “pure virtual” base class

# Interfaces

- Java allows for the creation of a type of absolute pure virtual class
- All methods in an interface are abstract (no code anywhere)
- Variables are rare but if there are any, they must be static or final
- Obviously, an interface cannot be instantiated
- Interface could be a reference to any object that “implements” the interface

# Default and static methods (JDK8+)

- Java recently added the capability to define methods inside an interface.  The methods must be labelled as “default” or “static” as appropriate.
- This is intended to add capability to existing interfaces without the need to update older classes’ that already implement the interface (without the new methods)

# A gas station example

- We want to design a GasStation class with a method called "fillErUp" which will refuel various things. The thing to be refuelled will be passed as a parameter to this method. What is the appropriate datatype for the parameter?
  - Car
  - Boat
  - Airplane
  - Vehicle (base class for the above)
  - Chainsaw
  - Refuelable (an interface)

# Cloneable

- A very common interface since all objects inherit the clone method from Object
- Implementing Cloneable allows for your object to be “cloned” (think copied).
- The clone method should be implemented but could simply call super.clone() – shallow copy.
- A deep copy could also be implemented in clone()

# Event Driven programming

- Unlike the imperative programming we've been doing since the beginning of CS-UY 1114, event driving programming doesn't follow a path from beginning to end.
- The user's interaction with the program dictates what happens next.
- By presenting the user with interactive elements (windows, buttons, etc) the user controls the flow of execution
- An action handler is a object who's method will be called when an event occurs.
- Action handlers must be registered for each interactive element in order to have activity performed on that interactive element

# AWT vs. Swing

- Java's first attempt at a window tool kit was "The Abstract Window Toolkit," java.awt. Much of it required too much code to implement and it was very complex.
- Java derived, in javax.swing (note the x), the Swing classes. They are compatible with many of the AWT components (because they're derived from them) but are much easier to use

# javax.swing.JFrame

- https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html
- The First class that you need to create.
- Responsible for "painting" the look and feel of a window
- Will "look" different on Windows, MacOS and Linux
- Constructor takes a "title" for the window
- Closing it doesn't, necessarily, end the program (it's an event, you handle it)

# javax.swing.JPanel

- A container to “hold” objects
- Can be manipulated (background can be changed)
- Can have a “layout”
- Must be added to the JFrame or another JPanel

# Layouts

- SquentialLayout – (the default)just adds components filling the space as it goes
- GridLayout – adds components to the container from top left to bottom right across rows.  Number of rows and columns is specified as is the spaces between them (hgap, vgap)
- BorderLayout –adds components in any order and uses defined constants for NORTH,EAST,SOUTH,WEST,CENTER to describe where the component belongs.

# JButton

- Can be clicked with a mouse
- Events are handled by an ActionListener which must be “added” to the object with addActionListener

# Action Listeners

- The ActionListener interface provides the easiest way to register a "callback" handler for when an action is performed.
- The actionPerformed method will be called when an action occurs (like a button pressed)