Computer Science: Create Your Own RPG Day #4

OPPTAG Explorations 2014

Brian Nakayama[1]

[1] Department of Computer Science, Iowa State University, Ames, IA 50010, USA

July 10th, 2014

Public vs Non-Public Classes

- Public: Visible to all classes in all packages. The name of the class must be the same as the name of the file.
- No Modifier: Visible only to classes in the same package. The name of the class does not have to be the same as the name of the file.

Public, Protected, and Private Variables and Methods

- Public: The variable *or method* can be seen by all classes in all packages.
- Private: The variable *or method* can only be seen by the class.
- Protected: The variable *or method* can be seen by all classes in the same package, and all *subclasses* (we will see this tomorrow).
- No Modifier: The variable *or method* can be seen by all classes in the same package, but not *subclasses* (we will see this tomorrow).

Easy Table for Memory

Modifier	Class	Package	Subclass	Other Packages
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Scoping it Out!

```
package package1;

public class class1 {
   int a;
   public double b;

   private void mult(int k, int l){
      a = k*l;
   }
}


class class2 {
   protected String c;
   private double d;
   public static short e;

   protected void setC(String m){
      this.c = m;
   }
}
```

```
package package2;

public class class3 {
   private boolean f;
   protected byte g;
   private static long h;

   protected void sayHi(){
      System.out.println("Hello World!");
   }
}

class class4 {
   public class3 i;
   static char j;
}
```

What is a GUI?

- A GUI is a *Graphical User Interface*.

What is a GUI?

- A GUI is a *Graphical User Interface*.
- Most applications you use on your computer, phone, or gaming device have a GUI.
- Are GUI's necessary? Can we play a 3D RTS with a GUI?

What is a GUI?

- A GUI is a *Graphical User Interface*.
- Most applications you use on your computer, phone, or gaming device have a GUI.
- Are GUI's necessary? Can we play a 3D RTS with a GUI?
- *Yes*. We don't necessarily need to see graphics to play a game. We could also use ASCII characters to approximate an image. (But that's no fun.)

What is a GUI?

- A GUI is a *Graphical User Interface*.
- Most applications you use on your computer, phone, or gaming device have a GUI.
- Are GUI's necessary? Can we play a 3D RTS with a GUI?
- *Yes*. We don't necessarily need to see graphics to play a game. We could also use ASCII characters to approximate an image. (But that's no fun.)
- In Java, most visible GUI objects are contained in the *Swing* package. Ways to interact with the GUI are kept in the *awt* package.

The Swing Package and Some Classes

- javax.swing.JFrame: This gives us the frame for a window.

The Swing Package and Some Classes

- javax.swing.JFrame: This gives us the frame for a window.
- javax.swing.JPanel: This gives us the body of our window.

The Swing Package and Some Classes

- javax.swing.JFrame: This gives us the frame for a window.
- javax.swing.JPanel: This gives us the body of our window.
- javax.swing.JTextArea: This gives us a 2D text field (square of text space).

The Swing Package and Some Classes

- javax.swing.JFrame: This gives us the frame for a window.
- javax.swing.JPanel: This gives us the body of our window.
- javax.swing.JTextArea: This gives us a 2D text field (square of text space).
- javax.swing.JTextField: This gives us a 1D text field (line of text space).

Example #1: Lets Make a Terminal!

Create a new class. Put the following in it.

```
private static JFrame jf = new JFrame("name");
private static JPanel jp = new JPanel(
    new BorderLayout());
private static JTextField textField= new JTextField(40);
private static JTextArea textArea= new JTextArea(10,40);
```

Example #1: Lets Make a Terminal!

Create a new class. Put the following in it.

```
private static JFrame jf = new JFrame("name");
private static JPanel jp = new JPanel(
    new BorderLayout());
private static JTextField textField= new JTextField(40);
private static JTextArea textArea= new JTextArea(10,40);
```

Create a main method in your class.

Example #1

Put the following in your main method. Then run your code to check if it works.

```
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
textArea.setEditable(false);
jp.add(textField, BorderLayout.NORTH);
jp.add(textArea, BorderLayout.SOUTH);
jf.add(jp);
jf.pack();
jf.setVisible(true);
```

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.
- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Causes the program to terminate when the window is closed.

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.
- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Causes the program to terminate when the window is closed.
- `new JTextField(<int>)`: Creates a text field that can hold `<int>` characters.

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.
- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Causes the program to terminate when the window is closed.
- `new JTextField(<int>)`: Creates a text field that can hold `<int>` characters.
- `new JTextArea(<rows>,<collums>)`: Creates a new text area. Each cell in the area can hold one character.

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.
- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Causes the program to terminate when the window is closed.
- `new JTextField(<int>)`: Creates a text field that can hold `<int>` characters.
- `new JTextArea(<rows>,<collums>)`: Creates a new text area. Each cell in the area can hold one character.
- `setEditable(false)`: Prevents the text area from being edited.

Example #1: Explanation

- `new JFrame(<text>)`: Creates a new window frame with the `text` on top.
- `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`: Causes the program to terminate when the window is closed.
- `new JTextField(<int>)`: Creates a text field that can hold `<int>` characters.
- `new JTextArea(<rows>,<collums>)`: Creates a new text area. Each cell in the area can hold one character.
- `setEditable(false)`: Prevents the text area from being edited.
- `new JPanel(new BorderLayout())`: Creates a layout. A *layout* is a way to organize objects in a panel.

Example #1: Explanation

- `jp.add(textField, BorderLayout.NORTH)`: Adds a text field towards the top of our panel.

Example #1: Explanation

- `jp.add(textField, BorderLayout.NORTH)`: Adds a text field towards the top of our panel.
- `jp.add(textArea, BorderLayout.SOUTH)`: Adds a text area towards the bottom of our panel.

Example #1: Explanation

- `jp.add(textField, BorderLayout.NORTH)`: Adds a text field towards the top of our panel.
- `jp.add(textArea, BorderLayout.SOUTH)`: Adds a text area towards the bottom of our panel.
- `jf.add(<JPanel>)`: Adds a panel to our frame.

Example #1: Explanation

- `jp.add(textField, BorderLayout.NORTH)`: Adds a text field towards the top of our panel.
- `jp.add(textArea, BorderLayout.SOUTH)`: Adds a text area towards the bottom of our panel.
- `jf.add(<JPanel>)`: Adds a panel to our frame.
- `jf.pack()`: Sets the size of the frame so that it is just big enough to hold our panel.

Example #1: Explanation

- `jp.add(textField, BorderLayout.NORTH)`: Adds a text field towards the top of our panel.
- `jp.add(textArea, BorderLayout.SOUTH)`: Adds a text area towards the bottom of our panel.
- `jf.add(<JPanel>)`: Adds a panel to our frame.
- `jf.pack()`: Sets the size of the frame so that it is just big enough to hold our panel.
- `jf.setVisible(<boolean>)`: Makes the frame visible (otherwise the user can't see it).

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.
- java.awt.event.KeyAdapter: This is a class that must be *extended*. It contains methods for key events.

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.
- java.awt.event.KeyAdapter: This is a class that must be *extended*. It contains methods for key events.
- java.awt.event.KeyEvent: This is an event for when someone uses the key board.

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.
- java.awt.event.KeyAdapter: This is a class that must be *extended*. It contains methods for key events.
- java.awt.event.KeyEvent: This is an event for when someone uses the key board.
- java.awt.event.WindowAdapter: This is also a class that must be *extended*. It contains methods for window events.

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.
- java.awt.event.KeyAdapter: This is a class that must be *extended*. It contains methods for key events.
- java.awt.event.KeyEvent: This is an event for when someone uses the key board.
- java.awt.event.WindowAdapter: This is also a class that must be *extended*. It contains methods for window events.
- java.awt.event.WindowEvent: This is an event for when someone closes, minimizes, or otherwise changes a window.

The Swing Package and Some Classes

- java.awt.BorderLayout: This is the layout that we used in the previous example.
- java.awt.event.KeyAdapter: This is a class that must be *extended*. It contains methods for key events.
- java.awt.event.KeyEvent: This is an event for when someone uses the key board.
- java.awt.event.WindowAdapter: This is also a class that must be *extended*. It contains methods for window events.
- java.awt.event.WindowEvent: This is an event for when someone closes, minimizes, or otherwise changes a window.

Example #2: Lets Interface with Our Terminal!

Create a new classfor your KeyAdapter:

```
class ka extends KeyAdapter{

   private JTextField textField;
   private JTextArea textArea;

   public ka(JTextField textField, JTextArea textArea){
      this.textField = textField;
      this.textArea = textArea;
   }
}
```

Example #2

Create the following method in your KeyAdapter:

```
@Override
public void keyPressed(KeyEvent e)
   switch(e.getKeyCode())
   case KeyEvent.VK_ENTER:
      if(!textField.getText().equals(""))
         String s = textArea.getText();
         textArea.setText(s + "\n"
               + textField.getText());
         textField.setText("");

   break;
```

Example #2: Explanation

- Add `textField.addKeyListener(new ka(textField, textArea));` in your main method to make the text field send Key Events.

Example #2: Explanation

- Add `textField.addKeyListener(new ka(textField, textArea));` in your main method to make the text field send Key Events.
- `@Override`: This is optional. It tells the compiler and the person reading the code that this method replaces the method of the same name in KeyAdapter.

Example #2: Explanation

- Add `textField.addKeyListener(new ka(textField, textArea));` in your main method to make the text field send Key Events.
- `@Override`: This is optional. It tells the compiler and the person reading the code that this method replaces the method of the same name in KeyAdapter.
- `keyPressed(KeyEvent e)`: This method is called whenever a person presses down on a key.

Example #2: Explanation

- Add `textField.addKeyListener(new ka(textField, textArea));` in your main method to make the text field send Key Events.
- `@Override`: This is optional. It tells the compiler and the person reading the code that this method replaces the method of the same name in KeyAdapter.
- `keyPressed(KeyEvent e)`: This method is called whenever a person presses down on a key.
- `e.getKeyCode()`: This method gets the code for the key pressed from the KeyEvent.

Example #2: Explanation

- Add `textField.addKeyListener(new ka(textField, textArea));` in your main method to make the text field send Key Events.
- `@Override`: This is optional. It tells the compiler and the person reading the code that this method replaces the method of the same name in KeyAdapter.
- `keyPressed(KeyEvent e)`: This method is called whenever a person presses down on a key.
- `e.getKeyCode()`: This method gets the code for the key pressed from the KeyEvent.
- `KeyEvent.VK_ENTER`: This is the event for a key being pressed.

The world Package

The most important classes:

- SimpleSolid: These are objects that can move, receive and give collisions (which calls the collision method). SimpleSolid must be *extended*.

The world Package

The most important classes:

- SimpleSolid: These are objects that can move, receive and give collisions (which calls the collision method). SimpleSolid must be *extended*.
- SimpleObject: These are objects that can move and receive collisions (they cannot collide with other SimpleObjects). SimpleObject must be *extended*.

The world Package

The most important classes:

- SimpleSolid: These are objects that can move, receive and give collisions (which calls the collision method). SimpleSolid must be *extended*.
- SimpleObject: These are objects that can move and receive collisions (they cannot collide with other SimpleObjects). SimpleObject must be *extended*.
- SimpleWorld: This manages all of the movements of objects, and drawing the images.

The world Package

The most important classes:

- SimpleSolid: These are objects that can move, receive and give collisions (which calls the collision method). SimpleSolid must be *extended*.
- SimpleObject: These are objects that can move and receive collisions (they cannot collide with other SimpleObjects). SimpleObject must be *extended*.
- SimpleWorld: This manages all of the movements of objects, and drawing the images.
- SimpleWorldObject: This object can draw over the screen (things like pause, dialog, etc.).

What Must Be Overridden

The following must be overridden in an object that *extends* SimpleSolid

- `abstract public void collision(SimpleObject s)`: This method is called automatically when a collision happens. The argument "s" is the object the solid had a collision with.
- `abstract public void update()`: This method is called 20 times per second! Use this to update the game's state.
- `abstract public char id()`: This method is used for other objects to find out the id of an object it is colliding with.

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).
- `player.setImage("girlN.png")`: changes the image of an object.

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).
- `player.setImage("girlN.png")`: changes the image of an object.
- `player.playSound("punch.wav")`: plays a sound (a midi or a wav).

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).
- `player.setImage("girlN.png")`: changes the image of an object.
- `player.playSound("punch.wav")`: plays a sound (a midi or a wav).
- `player.moveCell(x, y, true)`: moves the player to the cell (x, y) relative to its position.

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).
- `player.setImage("girlN.png")`: changes the image of an object.
- `player.playSound("punch.wav")`: plays a sound (a midi or a wav).
- `player.moveCell(x, y, true)`: moves the player to the cell (x, y) relative to its position.
- `player.moveCell(x, y, 8, false)`: over 8 frames, moves the player to the cell (x, y) in the world.

Solid Useful Methods

- `SimpleSolid player = new SimpleSolid("girlS.png")`: instantiates a solid with an image (preferably a png. Jpeg won't work.).
- `player.setImage("girlN.png")`: changes the image of an object.
- `player.playSound("punch.wav")`: plays a sound (a midi or a wav).
- `player.moveCell(x, y, true)`: moves the player to the cell (x, y) relative to its position.
- `player.moveCell(x, y, 8, false)`: over 8 frames, moves the player to the cell (x, y) in the world.
- `SimpleSolid personAbovePlayer = player.getNorthSolid()`: gets the object above the player.

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"
- `world.addSimpleObject(person, 5, 7)`: adds an object at the position ($x = 5$) and ($y = 7$). Returns true if the object was successfully added.

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"
- `world.addSimpleObject(person, 5, 7)`: adds an object at the position ($x = 5$) and ($y = 7$). Returns true if the object was successfully added.
- `world.removeSimpleObject(person)`: removes the object *person* if it's in the map. Returns true if the object was successfully removed.

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"
- `world.addSimpleObject(person, 5, 7)`: adds an object at the position ($x = 5$) and ($y = 7$). Returns true if the object was successfully added.
- `world.removeSimpleObject(person)`: removes the object *person* if it's in the map. Returns true if the object was successfully removed.
- `world.setSimpleWorldObject(swo)`: adds a simple world object.

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"
- `world.addSimpleObject(person, 5, 7)`: adds an object at the position ($x = 5$) and ($y = 7$). Returns true if the object was successfully added.
- `world.removeSimpleObject(person)`: removes the object *person* if it's in the map. Returns true if the object was successfully removed.
- `world.setSimpleWorldObject(swo)`: adds a simple world object.
- `world.setBGImage("floor.png")`: sets the background image for the world.

Worldly Useful Methods

- `SimpleWorld world = new SimpleWorld(20, 20, 16, 16, "My first java game!")`: Creates a map of 20x20 cells, each of which are 16x16 pixels wide. Gives the application the name "My first java game!"
- `world.addSimpleObject(person, 5, 7)`: adds an object at the position ($x = 5$) and ($y = 7$). Returns true if the object was successfully added.
- `world.removeSimpleObject(person)`: removes the object *person* if it's in the map. Returns true if the object was successfully removed.
- `world.setSimpleWorldObject(swo)`: adds a simple world object.
- `world.setBGImage("floor.png")`: sets the background image for the world.
- `world.start(false)`: starts the game either in fullscreen (true) or a window (false)

Sneak Peak #3

The following is in `SimpleWorld.start(false)`.
SimpleWorld extends the *JFrame* class.

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setTitle(title);
this.setIgnoreRepaint(true);
this.pack();
this.setResizable(false);
this.setVisible(true);
```

What do other languages look like?

- Similar to English, Java is just one language amongst many.

What do other languages look like?

- Similar to English, Java is just one language amongst many.
- However, most languages are all doing the same thing: they're all telling the computer how to compute. They just look different.

What do other languages look like?

- Similar to English, Java is just one language amongst many.
- However, most languages are all doing the same thing: they're all telling the computer how to compute. They just look different.
- Java is an *object* oriented, *imperative* language that runs on a *virtual machine*.
- Object oriented means that a language can create classes and objects, just as we've done in class so far.

What do other languages look like?

- Similar to English, Java is just one language amongst many.
- However, most languages are all doing the same thing: they're all telling the computer how to compute. They just look different.
- Java is an *object* oriented, *imperative* language that runs on a *virtual machine*.
- Object oriented means that a language can create classes and objects, just as we've done in class so far.
- Imperative means that our programs has statements and states (variables). If a programming language is imperative, we can change states after we've initialized them.

What do other languages look like?

- Similar to English, Java is just one language amongst many.
- However, most languages are all doing the same thing: they're all telling the computer how to compute. They just look different.
- Java is an *object* oriented, *imperative* language that runs on a *virtual machine*.
- Object oriented means that a language can create classes and objects, just as we've done in class so far.
- Imperative means that our programs has statements and states (variables). If a programming language is imperative, we can change states after we've initialized them.
- The *Java Virtual Machine* is a virtual program that simulates a computer inside your computer. Instead of creating bytecode for your computer, we compile our code into bytecode for the virtual machine instead.

Some languages are similar to Java.

- C# is almost identical.

```
class Program{

public static int Fibonacci(int n){
    int a = 0;
    int b = 1;
    // A comment.
    for (int i = 0; i < n; i++){
        int temp = a;
        a = b;
        b = temp + b;
    }
    return a;
}
```

Not all languages use a VM.

- Objective-C and C++ usually compile to machine binaries.

```
import <Foundation/Foundation.h>

int main (int argc, const char* argv[])
{
    NSAutoreleasePool *pool =
          [[NSAutoreleasePool alloc] init];
    NSLog (@"Hello, World!");
    [pool drain];
    return 0;
}
```

Not all languages are object oriented.

- C and FORTRAN (pre-2003) don't have classes..

```
int main(void){
   char *str[] = { "first", "second", "third", 0 };
   char **w = str;

   while(*w){
      printf("%s\n", *w++);
   }

   return 0;
}
```

Not All Languages are Imperative

- Haskell and ML are functional languages.

```
module Main where

main :: IO ()
main = putStrLn "Hello, World!"

fibonacci :: Integer -> Integer
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = fibonacci (n-1) + fibonacci (n-2)
```

Some Languages are Both!

- Python! A crazy (yet interesting) language.

```
#imperative
def f(x) :
    return x ** 2

#functional
g = lambda x: x**2

#We don't have to declare variables!
for i in range[1,9]:
    string = "countdracula" + str(i)
```

Not All Languages make Programs!

- Verilog and VHDL are languages used to design hardware.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity and_gate is
    Port ( IN1 : in  STD_LOGIC;
           IN2 : in  STD_LOGIC;
           OUTPUT : out STD_LOGIC;
end and_gate;

architecture Behavioral of and_gate is
begin
    OUTPUT <= IN1 and IN2; -- 2 input AND gate
end Behavioral;
```

Scripting Languages

- HTML, Javascript, CSS, LaTeX...

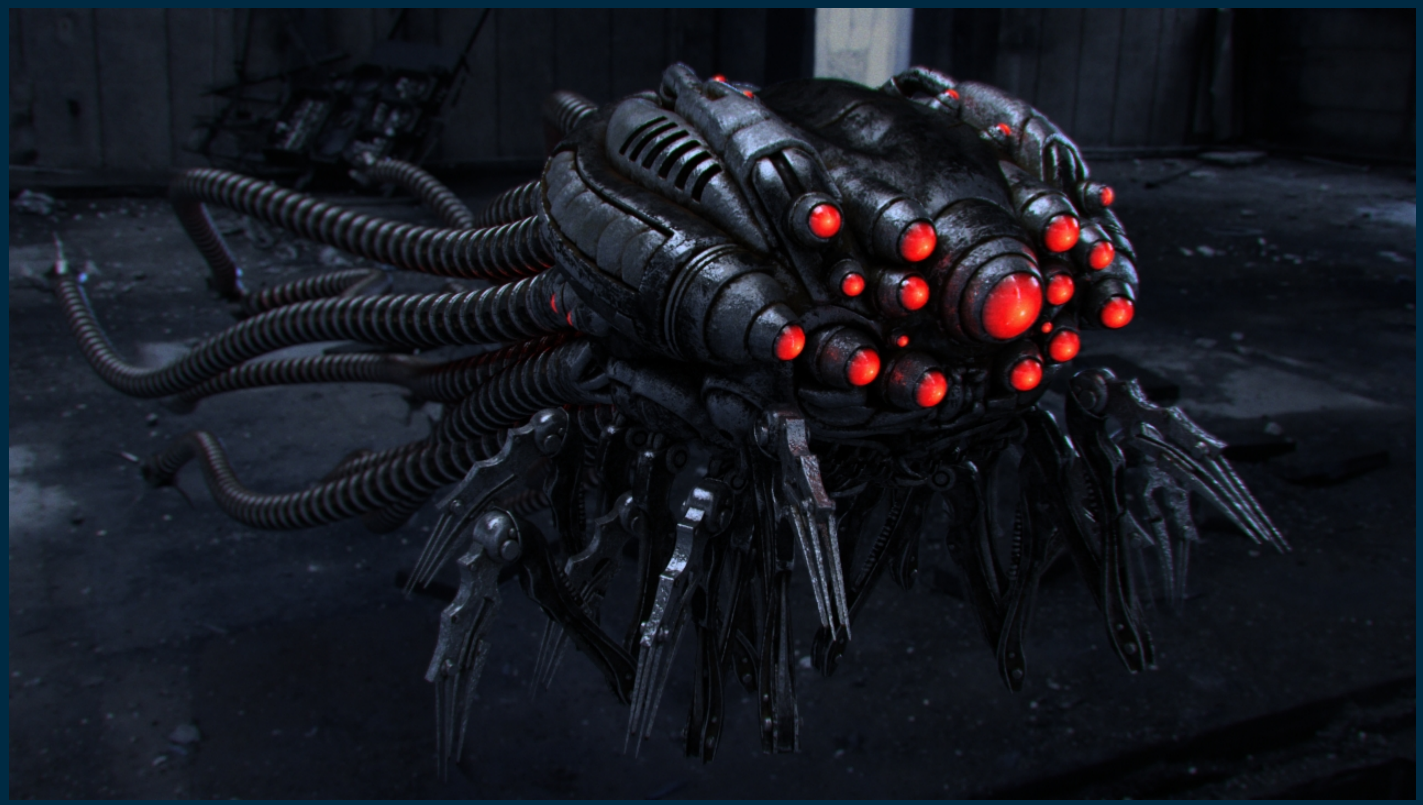
```
\begin{frame}[fragile]{Scripting Languages}
\begin{itemize}
\item HTML, Javascript, CSS, \LaTeX...
\end{itemize}
\begin{semiverbatim}\code{
    ...
}\end{semiverbatim}

\end{frame}
```

Assembly

- Language of the machines...

Assembly

- Language of the machines...



Assembly

- MIPS, ARM, AVR, x86...

Operation	$d = $s + $t;
Syntax	add $d, $s, $t
Encoding	0000 00ss ssst tttt dddd d000 0010 0000

Pop Quiz!

- What makes a language *imperative*?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.
- What makes a language *object oriented*?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.
- What makes a language *object oriented*?
- What is a *virtual machine*?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.
- What makes a language *object oriented*?
- What is a *virtual machine*?
- What language is almost identical to Java?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.
- What makes a language *object oriented*?
- What is a *virtual machine*?
- What language is almost identical to Java?
- Is java the best language?

Pop Quiz!

- What makes a language *imperative*?
- Is it true that all imperative languages are not functional?
- Name a scripting language.
- What makes a language *object oriented*?
- What is a *virtual machine*?
- What language is almost identical to Java?
- Is java the best language?
- Philosophical Question: Is there a way to *compile* the following into English:

안녕하세요 제 이름은 브라이언입니다

1's and 0's Everywhere!

- Electricity encoded as 1's and 0's are what's used by the computer.

1's and 0's Everywhere!

- Electricity encoded as 1's and 0's are what's used by the computer.
- Computers use billions of transistors which perform boolean logic.
- Boolean logic is just like the boolean operators we have learned (`&&`, `||`, `!`), except "1" is true and "0" is false.

1's and 0's Everywhere!

- Electricity encoded as 1's and 0's are what's used by the computer.
- Computers use billions of transistors which perform boolean logic.
- Boolean logic is just like the boolean operators we have learned (`&&`, `||`, `!`), except "1" is true and "0" is false.
- Using AND, OR, and NOT computers do everything you see on a daily basis.
- This means that colors, sounds, movies, everything can be represented by bits!

1's and 0's Everywhere!

- Electricity encoded as 1's and 0's are what's used by the computer.
- Computers use billions of transistors which perform boolean logic.
- Boolean logic is just like the boolean operators we have learned (`&&`, `||`, `!`), except "1" is true and "0" is false.
- Using AND, OR, and NOT computers do everything you see on a daily basis.
- This means that colors, sounds, movies, everything can be represented by bits!

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0
Position	5	4	3	2	1	0

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	0^*32	1^*16	0^*8	1^*4	1^*2	0^*1

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	0^*32	1^*16	0^*8	1^*4	1^*2	0^*1

$$\text{Total} = 0 + 16 + 0 + 4 + 2 + 0 = 22$$

But how do we read them?

- Bits can represent integers (`int`).
- To convert a binary number to an integer we must multiply each binary number by 2 to the power of its position (2^p), and then add them together.

Binary	0	1	0	1	1	0
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	0^*32	1^*16	0^*8	1^*4	1^*2	0^*1

$$\text{Total} = 0 + 16 + 0 + 4 + 2 + 0 = 22$$

$$010110 \text{ base } 2 = 22 \text{ base } 10$$

A few more examples

Binary	1	0	0	1

A few more examples

Binary	1	0	0	1
Position	3	2	1	0

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

1001 base 2 = 9 base 10

Binary	1	0	0	1	1	1

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

$$1001 \text{ base } 2 = 9 \text{ base } 10$$

Binary	1	0	0	1	1	1
Position	5	4	3	2	1	0

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

$$1001 \text{ base } 2 = 9 \text{ base } 10$$

Binary	1	0	0	1	1	1
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

$$1001 \text{ base } 2 = 9 \text{ base } 10$$

Binary	1	0	0	1	1	1
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	1^*32	0^*16	0^*8	1^*4	1^*2	1^*1

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

$$1001 \text{ base } 2 = 9 \text{ base } 10$$

Binary	1	0	0	1	1	1
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	1^*32	0^*16	0^*8	1^*4	1^*2	1^*1

$$\text{Total} = 32 + 0 + 0 + 4 + 2 + 1 = 39$$

A few more examples

Binary	1	0	0	1
Position	3	2	1	0
2^p	2^3	2^2	2^1	2^0
Multiplied	1^*8	0^*4	0^*2	1^*1

$$\text{Total} = 8 + 0 + 0 + 1 = 9$$

$$1001 \text{ base } 2 = 9 \text{ base } 10$$

Binary	1	0	0	1	1	1
Position	5	4	3	2	1	0
2^p	2^5	2^4	2^3	2^2	2^1	2^0
Multiplied	1^*32	0^*16	0^*8	1^*4	1^*2	1^*1

$$\text{Total} = 32 + 0 + 0 + 4 + 2 + 1 = 39$$

$$100111 \text{ base } 2 = 39 \text{ base } 10$$

Converting from Integers to Binary

To go backwards we need to find the highest power of 2 that is less than our number. This will be a '1'. We then subtract the highest power of 2 from our number, and then repeat.

13 base 10 = ?

Position	3	2	1	0

Converting from Integers to Binary

To go backwards we need to find the highest power of 2 that is less than our number. This will be a '1'. We then subtract the highest power of 2 from our number, and then repeat.

13 base 10 = ?

Position	3	2	1	0
Multiplied	1*8	?*4		

Converting from Integers to Binary

To go backwards we need to find the highest power of 2 that is less than our number. This will be a '1'. We then subtract the highest power of 2 from our number, and then repeat.

13 base 10 = ?

Position	3	2	1	0
Multiplied	1*8	0*4	?*2	
Binary	1	1	?	

Converting from Integers to Binary

To go backwards we need to find the highest power of 2 that is less than our number. This will be a '1'. We then subtract the highest power of 2 from our number, and then repeat.

13 base 10 = ?

Position	3	2	1	0
Multiplied	1^*8	0^*4	1^*2	$?^*1$
Binary	1	1	0	?

Converting from Integers to Binary

To go backwards we need to find the highest power of 2 that is less than our number. This will be a '1'. We then subtract the highest power of 2 from our number, and then repeat.

13 base 10 = 1011 base 2

Position	3	2	1	0
Multiplied	1^*8	0^*4	1^*2	1^*1
Binary	1	1	0	1

Quiz: One with the Machines!

- 15 base 10 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 = 3 base 10
- 010 base 2 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 = 3 base 10
- 010 base 2 = 2 base 10
- 1011 base 2 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 = 3 base 10
- 010 base 2 = 2 base 10
- 1011 base 2 = 11 base 10
- 01100 base 2 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 = 3 base 10
- 010 base 2 = 2 base 10
- 1011 base 2 = 11 base 10
- 01100 base 2 = 12 base 10
- 1011011 base 10 =

Quiz: One with the Machines!

- 15 base 10 = 1111 base 2
- 8 base 10 = 1000 base 2
- 5 base 10 = 101 base 2
- 42 base 10 = 101010 base 2
- 11 base 2 = 3 base 10
- 010 base 2 = 2 base 10
- 1011 base 2 = 11 base 10
- 01100 base 2 = 12 base 10
- 1011011 base 10 = 91 base 10

Game Engine Activites!

Lets make a maze game together! If you have not done so already, download the game engine (resources) from my website.