CSc 165
Computer Game Architecture

7 - Scripting

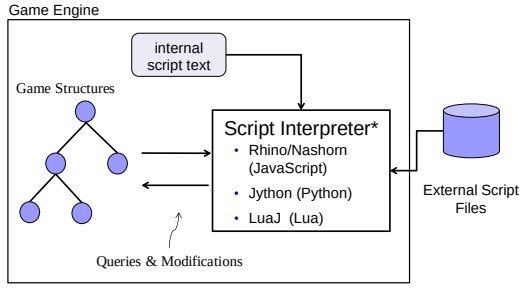
Overview

- **Scripting Concepts**
- **Script Interpreters ("Engines")**
- **Scripting Languages**
  - **JavaScript Basics**
- **Communicating with Scripts**
- **Using Scripts in Games**
- **Additional Scripting Engines**

Scripting

- Using external code to alter game world structure or game play
- Common "scripting languages":
  - JavaScript, Python, Lua
  - Others (Tcl, Scheme, Ruby, Smalltalk, VB…)
- Scripts often need access to game objects
  - Or at least, to a API
- Requires embedding an *interpreter* in game/game engine

Script Interpreters



*Also known as a Script *Engine*

Using a script engine in Java

- Get the Java script engine manager
- Use it to get the desired script engine*
- Use `eval(…)` to run the script interpreter
  - `eval(`*String*`)`, or
  - `eval(`*FileReader*`)`
- Scripts can also be compiled

(*) *be careful not to confuse the script engine with the game engine!*

JavaScript Basics

Comments
- Same as Java: // or /*… */ (no JavaDoc /** … */ form)

Variables
- Declared with 'var' (optional)
- Either *global* or *local* (inside a function) – no "class scope"
- Syntax: same as Java (e.g. start with letter or "_")
  - Cannot use reserved words (most Java reserved words, plus others)
- "weakly typed" – type determined by assigned value

```
var i = 8;                          // i is an int
var pi = 3.14159;                   // pi is a real
var j = "Hello";                    // j is a string
var k = 42 + " is the answer";      // k is also a string
var m = i < 10;                     // m is a Boolean = true
```

JavaScript Basics (cont.)

Operators

same as Java (+, -, *, /, %, ==, !=, <, >, <=, >=, &&, ||, !, =)

Control statements

```
if (cond) {...} else {...}      // same as Java
for (var i=0; i<3; i++) {...}   // almost Java
while (cond) {...}              // same as Java
try {...} catch(e){...}         // same as Java
```

Functions

- Global scope by default
- Defined with keyword: **`function`**

Communicating with Scripts

The Java Scripting API: **`javax.script.*`**

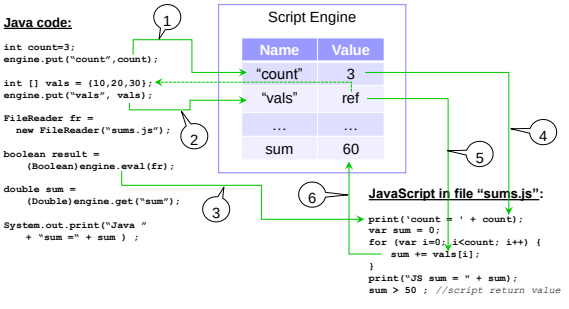
Allows Java to:

- Pass data into a script: **`engine.put()`**
- Get data back from a script: **`engine.get()`**
  - Scripts can assign values to vars accessible by Java
  - Scripts also have a "return value"

Allows scripts to:

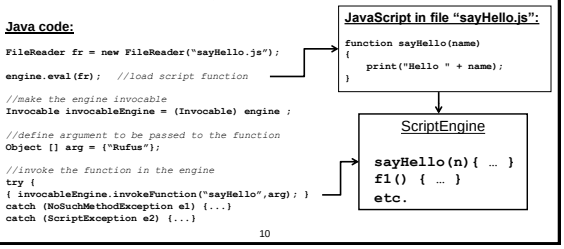
- Get data from Java
- Pass data to Java
- Invoke methods in Java objects

Java/Script Communication



Invoking Script Functions

- Define script function
- Load function into engine (using **`eval()`**)
- Cast engine as an "Invocable" object
- Use **`invokeFunction()`** to call function



Uses for scripting

- Gameworld creation and initialization
- Dynamically modifying game details
- Providing user-defined functions that can be called from a Java application
- Modifying player and non-player characters
- Modifying game features
- Testing

Using Other Script Engines

Lua

```
//get the Lua engine
ScriptEngine luaEngine = factory.getEngineByExtension(".lua");
//insert variable "x" with value 25
luaEngine.put("x", 25);
//run a Lua script to compute a function of x
try
{  luaEngine.eval("y = math.sqrt(x)"); }
catch (ScriptException e)
{  System.out.println(e); }
System.out.println("Hello Lua: " + "y=" + luaEngine.get("y"));
```

Python

```
// construct a Python script
String lsep = System.getProperty("line.separator");
String a = "import sys" + lsep;
       a += "import java.io as javaio" + lsep;
       a += "currentdir = javaio.File (\".\")" + lsep;
       a += "print \"Hello Python: Current directory : \" +
                  currentdir.getCanonicalPath()" + lsep;
//get the Python engine (Jython)
ScriptEngine pythonEngine = factory.getEngineByExtension("py");
//run the Python script
try
{  pythonEngine.eval(a); }
catch (ScriptException e)
{  System.out.println(e); }
```

Additional JavaScript Features

Arrays

- Size defined by *parentheses* at declaration

```
var foo = new Array(10);
var bar = new Array(5);
```

- Indexing from zero and using brackets (like Java)

```
foo[0] = 42;   bar[4] = 99.9;
```

- Mixed element types allowed

```
var stuff = new Array ("a string", 12, 98.6, true);
```

- Dynamically resizeable

```
var colors = new Array();      //colors has no elements
colors[2] = "red" ;            //colors now has 3 elements
                               //  ( [0] and [1] == null )
```

- Properties and methods

```
length,  indexOf(),  concat(), toString(), ...
```


Additional JavaScript Features (cont.)

Built-in Objects :

```
var currentTime = new Date();
var month = currentTime.getMonth() + 1;
var day = currentTime.getDate();
var year = currentTime.getFullYear();
```

User-created Objects :

```
var personObj=new Object();
personObj.firstname="John";       //properties ("fields") are
personObj.lastname="Doe";         //  created when defined
personObj.age=50;
personObj.eyecolor="blue";
```


Additional JavaScript Features (cont.)

User-defined Object Constructors:

```
function person(firstname,lastname,age,eyecolor)
{ this.firstname = firstname;
  this.lastname = lastname;
  this.age = age;
  this.eyecolor = eyecolor;
  this.newLastName = newLastName;        //method invocation
}

function newLastName(new_lastname)
{ this.lastname = new_lastname;
}

var myFather = new person("John","Doe",50,"blue");
var myMother = new person("Sally","Rally",48,"green");
myMother.newLastName("Doe");
```


Additional JavaScript Features (cont.)

User-defined Object Constructors (another example):

```
//object creation function
function circle(r)
{ this.radius = r;                    //radius property
  this.area = getArea;                //function invocation
  this.diameter = getDiameter;        //function invocation
}

function getArea()                    //function definition
{ var area = this.radius*this.radius*3.14;
  return area;
}

function getDiameter()                //function definition
{ var d = this.radius*2;
  return d;
}

var myCircle = new circle(20);
print("area = " + myCircle.area());   //print is a Nashorn method
print("diameter = " + myCircle.diameter());
```