**Question 1: DLL**

// math.cpp

// creates math.dll

#include <Windows.h>

BOOL APIENTRY DLLMain(HANDLE hModule, DWORD dwReason, LPVOID LpReserved)

{

return true;

}

extern "C" \_declspec(dllexport)

float Addition(float x, float y)

{

return(x + y);

}

extern "C" \_declspec(dllexport)

float Subtraction(float x, float y)

{

return(x - y);

}

extern "C" \_declspec(dllexport)

float Multiplication(float x, float y)

{

return(x \* y);

}

extern "C" \_declspec(dllexport)

float Division(float x, float y)

{

return(x / y);

}

// main.cpp

// driver for math.dll

#include <stdio.h>

#include <Windows.h>

#include <iostream>

using namespace std;

typedef float (\*importFunction) (float, float);

int main()

{

importFunction Addition, Subtraction, Multiplication, Division;

float num1, num2;

float sum, diff, prod, quot;

HINSTANCE hinstLib = LoadLibrary(TEXT("Math.dll"));

Addition = (importFunction) GetProcAddress(hinstLib, "Addition");

Subtraction = (importFunction) GetProcAddress(hinstLib, "Subtraction");

Multiplication = (importFunction) GetProcAddress(hinstLib, "Multiplication");

Division = (importFunction) GetProcAddress(hinstLib, "Division");

cout << "Enter two numbers\n";

cin >> num1 >> num2;

sum = Addition(num1, num2);

diff = Subtraction(num1, num2);

prod = Multiplication(num1, num2);

quot = Division(num1, num2);

cout << "Sum: " << sum

<< "\nDifference: " << diff

<< "\nProduct: " << prod

<< "\nQuotient: " << quot;

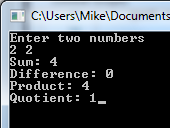
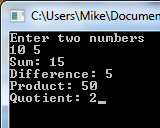
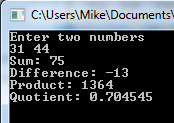
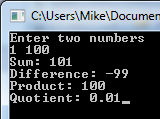
FreeLibrary(hinstLib);

return 0;

}

Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functions\Input** | **2 2** | **10 5** | **31 44** | **1 100** |
| **Addition** | 4 | 15 | 75 | 101 |
| **Subtraction** | 0 | 5 | -13 | -99 |
| **Multiplication** | 4 | 50 | 1364 | 100 |
| **Division** | 1 | 2 | 0.704545 | 0.01 |

**   **

**Question 2: Java**

(1)

// Employee.java

// Defines the class Employee

**import** java.text.NumberFormat;

**public** **class** Employee {

**private** String name;

**private** **int** numA;

**private** **char** numB;

**private** String hireDate;

NumberFormat form = NumberFormat.*getInstance*();

**public** Employee(String n, **int** a, **char** b, String hd) // Constructor

{

name = n;

numA = a;

numB = b;

hireDate = hd;

form.setMinimumIntegerDigits(3);

}

**public** Employee() // Default constructor

{

name = "";

numA = 0;

numB = ' ';

hireDate = "";

form.setMinimumIntegerDigits(3);

}

**public** String getName()

{

**return** name;

}

**public** String getNum()

{

**return** "" + form.format(numA) + '-' + numB;

}

**public** String getHireDate()

{

**return** hireDate;

}

**public** **void** setName(String n)

{

name = n;

}

**public** **void** setNum(**int** a, **char** b)

{

numA = a;

numB = b;

}

**public** **void** setHireDate(String hd)

{

hireDate = hd;

}

}

(2)

// ProductionWorker.java

// Defines ProductionWorker class

**public** **class** ProductionWorker **extends** Employee{

**private** **int** shift;

**private** **double** rate;

**public** ProductionWorker(String name, **int** a, **char** b,

String hd, **int** sft, **double** rt)

{

setName(name);

setNum(a, b);

setHireDate(hd);

setShift(sft);

setRate(rt);

}

**public** ProductionWorker()

{

setName("");

setNum(1, ' ');

setHireDate("");

setShift(1);

setRate(0);

}

**public** **int** getShift()

{

**return** shift;

}

**public** **double** getRate()

{

**return** rate;

}

**public** **void** setShift(**int** sft)

{

shift = sft;

}

**public** **void** setRate(**double** rt)

{

rate = rt;

}

}

(2)

// TestEmployee.java

// Driver for Employee and ProductionWorker classes

**public** **class** TestEmployee {

**public** **static** **void** printPW(ProductionWorker pw)

{

System.*out*.printf("\n%s: %s\n%s: %s\n%s: %s\n%s: %d\n%s: $%.2f%n\n",

"Name", pw.getName(),

"Employee Number", pw.getNum(),

"Hire Date", pw.getHireDate(),

"Shift", pw.getShift(),

"Pay Rate", pw.getRate());

}

**public** **static** **void** main(String args[])

{

ProductionWorker scott = **new** ProductionWorker(

"Scott Pilgrim", 999, 'Q', "4/23/1982", 2, 7.00);

ProductionWorker gman = **new** ProductionWorker();

gman.setName("Gideon Graves");

gman.setNum(1, 'G');

gman.setHireDate("12/25/81");

gman.setShift(1);

gman.setRate(20.00);

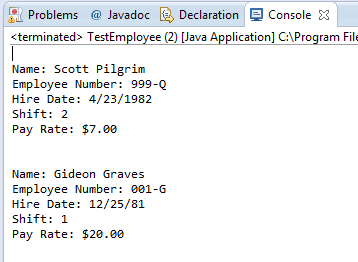
*printPW*(scott);

*printPW*(gman);

}

}

Driver Output:

I tested the classes using two objects of the ProductionWorker class. I gave them both all valid data to test its storage and output.

For the worker "Scott Pilgrim", I used the constructor. For "Gideon Graves" I used the default constructor, then the mutator functions to assign values.

Changes to Employee.java

**public** **void** setNum(**int** a, **char** b)

{

**try**{

numA = a;

**if**(numA < 0 || numA > 999)

**throw** **new** InvalidEmployeeNumber();

}

**catch**(InvalidEmployeeNumber e)

{

numA = 0;

}

numB = b;

}

**public** **void** setHireDate(String hd)

{

hireDate = hd;

}

**class** InvalidEmployeeNumber **extends** Exception

{

**public** InvalidEmployeeNumber()

{

System.*out*.println("Invalid Employee Number");

}

}

Changes to ProductionWorker.java

**public** **void** setShift(**int** sft) // edited to include exception handling

{

**try**{

shift = sft;

**if**(shift != 1 && shift != 2)

**throw** **new** InvalidShift();

}

**catch**(InvalidShift e)

{

shift = 1;

}

}

**public** **void** setRate(**double** rt) // edited to include exception handling

{

**try**{

rate = rt;

**if**(rate < 0)

**throw** **new** InvalidPayrate();

}

**catch**(InvalidPayrate e)

{

rate = 0;

} }

**class** InvalidShift **extends** Exception

{

**public** InvalidShift()

{

System.*out*.println("Invalid Shift");

}

}

**class** InvalidPayrate **extends** Exception

{

**public** InvalidPayrate()

{

System.*out*.println("Invalid Payrate");

}

}

// TestEmployee.java

// Driver for exception handling Employee and ProductionWorker classes

**public** **class** TestEmployee {

**public** **static** **void** printPW(ProductionWorker pw)

{

System.*out*.printf("\n%s: %s\n%s: %s\n%s: %s\n%s: %d\n%s: $%.2f%n\n",

"Name", pw.getName(),

"Employee Number", pw.getNum(),

"Hire Date", pw.getHireDate(),

"Shift", pw.getShift(),

"Pay Rate", pw.getRate());

}

**public** **static** **void** main(String args[])

{

ProductionWorker scott = **new** ProductionWorker(

"Scott Pilgrim", 1000, 'Q', "4/23/1982", 1, -7.00);

ProductionWorker gman = **new** ProductionWorker();

gman.setName("Gideon Graves");

gman.setNum(1, 'G');

gman.setHireDate("12/25/81");

gman.setShift(5);

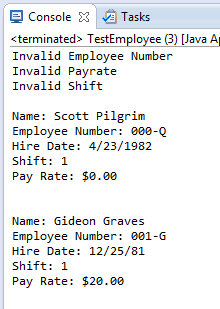
gman.setRate(20.00);

*printPW*(scott);

*printPW*(gman);

}

}

Driver Output:

For this test, worker "Scott Pilgrim" receives an invalid employee number (1000), and an invalid pay rate (-7.00). "Gideon Graves" is given an invalid shift number (5). The program throws an exception for each invalid entry and gives the assigns default values.

(3)

Changes to ProductionWorker.java

**public** **class** ProductionWorker **extends** Employee **implements** Sortable

**public** **boolean** greaterThan(Sortable y)

{

**if**(rate > ((ProductionWorker) y).getRate())

**return** **true**;

**else**

**return** **false**;

}

// Sortable.java

// Creates an interface Sortable

**public** **interface** Sortable

{

**public** **boolean** greaterThan(Sortable y);

}

// TestEmployee.java

// Driver for interface addition to ProductionWorker class

**public** **class** TestEmployee {

**public** **static** **void** printPW(ProductionWorker pw)

{

System.*out*.printf("\n%s: %s\n%s: %s\n%s: %s\n%s: %d\n%s: $%.2f%n\n",

"Name", pw.getName(),

"Employee Number", pw.getNum(),

"Hire Date", pw.getHireDate(),

"Shift", pw.getShift(),

"Pay Rate", pw.getRate());

}

**public** **static** **void** main(String args[])

{

ProductionWorker workers[] = **new** ProductionWorker[3];

workers[0] = **new** ProductionWorker(

"Poory McPoor", 0, 'P', "12/12/12", 2, 20);

workers[1] = **new** ProductionWorker(

"Middle McMan", 50, 'M', "6/6/6", 1, 25);

workers[2] = **new** ProductionWorker(

"Richie McRich", 100, 'R', "1/1/1", 1, 30);

ProductionWorker high = **new** ProductionWorker();

**for**(**int** i=0; i<3; i++)

**if**(workers[i].greaterThan(high))

high = workers[i];

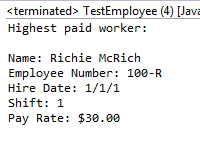
System.*out*.println("Highest paid worker:");

*printPW*(high);

}

}

Driver Output:



This driver creates a three item array of ProductionWorker objects. It gives them all valid data, and assigns pay rates of 20, 25, and 30. It then loops through the array and uses greaterThan function to find the worker with the highest pay and outputs it.

**Question 3: C#**

(1)

// Employee.cs

// Defines Employee class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_Part\_1

{

class Employee

{

private string name;

private int numA;

private char numB;

private string hireDate;

// NumberFormat form = NumberFormat.getInstance();

public Employee(String n, int a, char b, String hd)

{

name = n;

numA = a;

numB = b;

hireDate = hd;

//form.setMinimumIntegerDigits(3);

}

public Employee()

{

name = "";

numA = 0;

numB = ' ';

hireDate = "";

//form.setMinimumIntegerDigits(3);

}

public string getName()

{

return name;

}

public string getNum()

{

return String.Format("{0:000}", numA) + '-' + numB;

}

public string getHireDate()

{

return hireDate;

}

public void setName(string n)

{

name = n;

}

public void setNum(int a, char b)

{

numA = a;

numB = b;

}

public void setHireDate(string hd)

{

hireDate = hd;

}

}

}

(2)

// ProductionWorker.cs

// Defines ProductionWorker class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_Part\_1

{

class ProductionWorker : Employee

{

private int shift;

private double rate;

public ProductionWorker(String name, int a, char b,

String hd, int sft, double rt)

{

setName(name);

setNum(a, b);

setHireDate(hd);

setShift(sft);

setRate(rt);

}

public ProductionWorker()

{

setName("");

setNum(0, ' ');

setHireDate("");

setShift(1);

setRate(0);

}

public int getShift()

{

return shift;

}

public double getRate()

{

return rate;

}

public void setShift(int sft)

{

shift = sft;

}

public void setRate(double rt)

{

rate = rt;

}

}

}

// TestEmployee.cs

// Driver for Employee and ProductionWorker classes

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_Part\_1

{

class TestEmployee

{

static void Main(string[] args)

{

ProductionWorker scott = new ProductionWorker(

"Scott Pilgrim", 999, 'Q', "4/23/1982", 2, 7.00);

ProductionWorker gman = new ProductionWorker();

gman.setName("Gideon Graves");

gman.setNum(1, 'G');

gman.setHireDate("12/25/81");

gman.setShift(1);

gman.setRate(20.00);

Console.WriteLine("Name: " + scott.getName());

Console.WriteLine("Employee Number: " + scott.getNum());

Console.WriteLine("Hire Date: " + scott.getHireDate());

Console.WriteLine("Shift: " + scott.getShift());

Console.WriteLine("Pay Rate: " + scott.getRate().ToString("0.00"));

Console.WriteLine("\nName: " + gman.getName());

Console.WriteLine("Employee Number: " + gman.getNum());

Console.WriteLine("Hire Date: " + gman.getHireDate());

Console.WriteLine("Shift: " + gman.getShift());

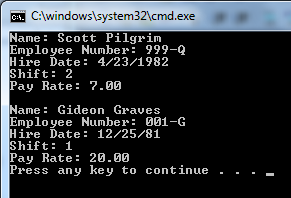
Console.WriteLine("Pay Rate: " + gman.getRate().ToString("0.00"));

}

}

}

**Driver Output:**

****This driver creates two objects of the ProductionWorker class. Both of them contain all valid data. Worker "Scott Pilgrim" was created and defined in the constructor, while "Gideon Graves" was created with the default constructor, then assigned data through the mutator functions.

(3)

**Changes to Employee.cs**

public void setNum(int a, char b)

{

try

{

numA = a;

if (numA < 0 || numA > 999)

throw new InvalidEmployeeNumber();

}

catch(InvalidEmployeeNumber)

{

numA = 0;

}

numB = b;

}

class InvalidEmployeeNumber : Exception

{

public InvalidEmployeeNumber()

{

Console.WriteLine("Invalid Employee Number");

}

}

**Changes to ProductionWorker.cs**

public void setShift(int sft)

{

try

{

shift = sft;

if (shift != 1 && shift != 2)

throw new InvalidShift();

}

catch (InvalidShift)

{

shift = 1;

}

}

public void setRate(double rt)

{

try

{

rate = rt;

if (rate < 0)

throw new InvalidPayrate();

}

catch (InvalidPayrate)

{

rate = 0;

}

}

class InvalidShift : Exception

{

public InvalidShift()

{

Console.WriteLine("Invalid Shift");

}

}

class InvalidPayrate : Exception

{

public InvalidPayrate()

{

Console.WriteLine("Invalid Payrate");

}

}

// TestEmployee.cs

// Driver for exception handling classes

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_Part\_1

{

class TestEmployee

{

static void Main(string[] args)

{

ProductionWorker scott = new ProductionWorker(

"Scott Pilgrim", 1000, 'Q', "4/23/1982", 2, -7.00);

ProductionWorker gman = new ProductionWorker();

gman.setName("Gideon Graves");

gman.setNum(-600, 'G');

gman.setHireDate("12/25/81");

gman.setShift(12);

gman.setRate(20.00);

Console.WriteLine("Name: " + scott.getName());

Console.WriteLine("Employee Number: " + scott.getNum());

Console.WriteLine("Hire Date: " + scott.getHireDate());

Console.WriteLine("Shift: " + scott.getShift());

Console.WriteLine("Pay Rate: " + scott.getRate().ToString("0.00"));

Console.WriteLine("\nName: " + gman.getName());

Console.WriteLine("Employee Number: " + gman.getNum());

Console.WriteLine("Hire Date: " + gman.getHireDate());

Console.WriteLine("Shift: " + gman.getShift());

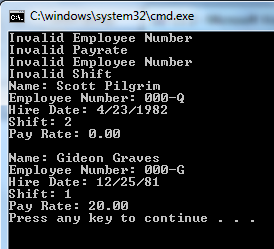
Console.WriteLine("Pay Rate: " + gman.getRate().ToString("0.00"));

}

}

}

Driver Output:

****

In this driver, worker "Scott Pilgrim" was given an invalid Employee Number (1000) and an invalid Pay rate (-7.00). "Gideon Graves" was given an invalid employee number (-600), and an invalid shift (12). As you can see, the exception handlers caught two invalid employee numbers, and one each of payrate and shift errors. These fields were then assigned default values.

(4)

Changes to ProductionWorker.cs

public void setShift(int sft)

{

try

{

shift = sft;

if (shift != 1 && shift != 2)

throw new InvalidShift();

}

catch (InvalidShift)

{

shift = 1;

}

}

public void setRate(double rt)

{

try

{

rate = rt;

if (rate < 0)

throw new InvalidPayrate();

}

catch (InvalidPayrate)

{

rate = 0;

}

}

public bool greaterThan(Sortable y)

{

if (getRate() > ((ProductionWorker)y).getRate())

return true;

return false;

}

// Sortable.cs

// Initiates interface Sortable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_3

{

interface Sortable

{

bool greaterThan(Sortable y);

}

}

// TestEmployee.cs

// Driver for interface implementation

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_Part\_1

{

class TestEmployee

{

static void Main(string[] args)

{

ProductionWorker[] workers = new ProductionWorker[3];

workers[0] = new ProductionWorker(

"Poory McPoor", 0, 'P', "12/12/12", 2, 20);

workers[1] = new ProductionWorker(

"Middle McMan", 50, 'M', "6/6/6", 1, 25);

workers[2] = new ProductionWorker(

"Richie McRich", 100, 'R', "1/1/1", 1, 30);

ProductionWorker high = new ProductionWorker();

for(int i=0; i<3; i++)

if(workers[i].greaterThan(high))

high = workers[i];

Console.WriteLine("\nName: " + high.getName());

Console.WriteLine("Employee Number: " + high.getNum());

Console.WriteLine("Hire Date: " + high.getHireDate());

Console.WriteLine("Shift: " + high.getShift());

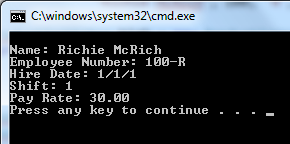
Console.WriteLine("Pay Rate: " + high.getRate().ToString("0.00"));

}

}

}

Driver Output:



This driver creates a three item array of ProductionWorker objects. It gives them all valid data, and assigns pay rates of 20, 25, and 30. It then loops through the array and uses greaterThan function to find the worker with the highest pay and outputs it.

**Conclusion:**

This project very well demonstrates how similar Java and C# are. My class definitions and implementations between them are nearly identical with only a few minor tweaks to the syntax. The only notable functional difference I came across was Java's support of number formatting options that I could not find in C#.

Lastly, programming the C++ file to create a DLL was fairly simple, though difficult to bug test for runtime miscalculations.