**Engine**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace C# 7

{

classEngine

{

privateint cylinders;

privateint horsepower;

// no-argument constructor

public Engine()

{

cylinders = 0;

horsepower = 0;

}

// property of cylinder

publicint Cylinders

{

set

{

cylinders = value;

}

get

{

return cylinders;

}

}

// property for horsepower

publicint HorsePower

{

set

{

horsepower = value;

}

get

{

return horsepower;

}

}

// overridden tostring method

// returns the state of the object

publicoverrideString ToString()

{

returnString.Format("Engine Cylinders\t: {0}\n" +

"Engine Horsepower\t: {1}\n", cylinders, horsepower);

}

// reads data from the console

publicvoid ReadRacerData()

{

Console.Write("Enter number of cylinders: ");

cylinders = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter horsepower of racer's engine: ");

horsepower = Convert.ToInt32(Console.ReadLine());

}

}

}

**Hot Rod**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace C# 7B

{

classHotRod : Racer

{

privatebool blower;

// default constructor

public HotRod()

: base()

{

blower = false;

}

// 4-argument constructor

public HotRod(String name, int speed, Engine engine, bool blower)

: base(name, speed, engine)

{

this.blower = blower;

}

// public property for blower attribute

publicbool Blower

{

set

{

blower = value;

}

get

{

return blower;

}

}

// implemenatin of IsDead method

publicoverridebool IsDead()

{

Random rnd = newRandom();

Boolean dead;

if (Speed > 50 && rnd.NextDouble() > 0.6)

{

if (Eng.HorsePower < 300 && blower == true)

dead = false;

else

dead = true;

}

elseif (Speed > 100 && rnd.NextDouble() > 0.4)

{

if (Eng.HorsePower >= 300 && blower == true)

dead = true;

else

dead = false;

}

else

dead = false;

return dead;

}

// returns state of the object

publicoverrideString ToString()

{

returnString.Format("{0} " +

"Racer's Type\t\t: {1}\n" +

"With Blower Option\t: {2}\n" +

"Still Working?\t\t: {3}\n", base.ToString(), this.GetType().Name.ToUpper(), blower ? "Yes" : "No", IsDead() ? "No!" : "Yes!");

}

// reads required value from the console

publicoverridevoid ReadRacerData()

{

bool valid = false;

base.ReadRacerData();

string inputString = string.Empty;

char value = 'u';

do

{

Console.Write("Enter blower option (Y / N): ");

inputString = Console.ReadLine();

if (!(String.IsNullOrEmpty(inputString)))

{

valid = char.TryParse(inputString, out value);

}

if (!valid)

Console.WriteLine("Invalid option. Try again!");

} while (!valid);

if (value == 'Y' || value == 'y')

blower = true;

else

blower = false;

}

}

}

**Program**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace C# 7C

{

classProgram

{

staticvoid Main(string[] args)

{

Console.WriteLine("Welcome to Abstract Racer Inheritance Hierarchy Program");

Console.WriteLine("CIS247a, Week 7 iLab");

Console.WriteLine("Name: Kevin Nguyen \n");

Console.WriteLine("\nThis program implements a complete inheritance hierarchy using abstract \nclasses and dynamic binding.");

// Array of Racer objects

Racer[] racers = newRacer[2];

// HotRod object

racers[0] = newHotRod();

// StreetTuner object

racers[1] = newStreetTuner();

// reads data for the objects

CollectRacerInformation(racers[0]);

DisplayRacerInformation(racers[0]);

CollectRacerInformation(racers[1]);

DisplayRacerInformation(racers[1]);

Console.WriteLine("Thank you for using the Abstract Racer Inheritance Hierarchy");

Console.ReadKey();

}

staticvoid CollectRacerInformation(Racer racer)

{

// reads the data for the given racer object

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Collecting Racer's Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

racer.ReadRacerData();

}

staticvoid DisplayRacerInformation(Racer racer)

{

// displays the data of the given racer object

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Display Racer's Information \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Console.WriteLine(racer.ToString());

}

}

}

**Racer**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace C# 7D

{

abstractclassRacer

{

privateString name; // racer name

privateint speed; // racer speed

privateEngine eng; // racer engine

// default construtor

public Racer()

{

name = "NOT GIVEN";

speed = 0;

eng = newEngine();

} // 3- argument construtor

public Racer(String name, int speed, Engine engine)

{

this.name = name;

this.speed = speed;

this.eng = engine;

}

// property for racer name

publicString Name

{

set

{

name = value;

}

get

{

return name;

}

}

//property for racer's speed

publicint Speed

{

set

{

speed = value;

}

get

{

return speed;

}

}

// property for racer's engine

publicEngine Eng

{

set

{

eng = value;

}

get

{

return eng;

}

}

// abstract mehod isdead

publicabstractbool IsDead();

// overridden tostring method to display state of the object

publicoverrideString ToString()

{

returnString.Format("Racer's Name\t\t: {0}\n" +

"Racer's speed\t\t: {1}\n{2}",

name, speed, eng.ToString());

}

// to read required values from the console

publicvirtualvoid ReadRacerData()

{

Console.Write("Enter racer's name: ");

name = Console.ReadLine();

Console.Write("Enter racer's speed: ");

speed = Convert.ToInt32(Console.ReadLine());

eng.ReadRacerData();

}

}

}

**Street Tuner**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespaceCIS247\_WK7\_Lab\_NGUYEN

{

classStreetTuner : Racer

{

privatebool nitrous;

// default constructor

public StreetTuner()

: base()

{

nitrous = false;

}

// 4-argument construtor

public StreetTuner(String name, int speed, Engine engine, bool nitrous)

: base(name, speed, engine)

{

this.nitrous = nitrous;

}

// public property of nitruos variable

publicbool Nitrous

{

set

{

nitrous = value;

}

get

{

return nitrous;

}

}

// implemenation of IsDead method

publicoverridebool IsDead()

{

Random rnd = newRandom();

Boolean dead;

if (Speed > 50 && rnd.NextDouble() > 0.6)

{

if (Eng.HorsePower < 300 && nitrous == true)

dead = false;

else

dead = true;

}

elseif (Speed > 100 && rnd.NextDouble() > 0.4)

{

if (Eng.HorsePower >= 300 && nitrous == true)

dead = true;

else

dead = false;

}

else

dead = false;

return dead;

}

// ToString mehod to retutn state of the object

publicoverrideString ToString()

{

returnString.Format("{0}" +

"Racer's Type\t\t: {1}\n" +

"With Nitrus Option\t: {2}\n" +

"Still Working?\t\t: {3}\n", base.ToString(), this.GetType().Name.ToUpper(), nitrous ? "Yes" : "No", IsDead() ? "No!" : "Yes!");

}

// Reads the required attributes from the console

// overrides the base class method

publicoverridevoid ReadRacerData()

{

bool valid = false;

base.ReadRacerData();

string inputString = string.Empty;

char value = 'u';

do

{

Console.Write("Enter nitrous option (Y / N): ");

inputString = Console.ReadLine();

if (!(String.IsNullOrEmpty(inputString)))

{

valid = char.TryParse(inputString, out value);

}

if (!valid)

Console.WriteLine("Invalid option. Try again!");

} while (!valid);

if (value == 'Y' || value == 'y')

nitrous = true;

else

nitrous = false;

}

}

}