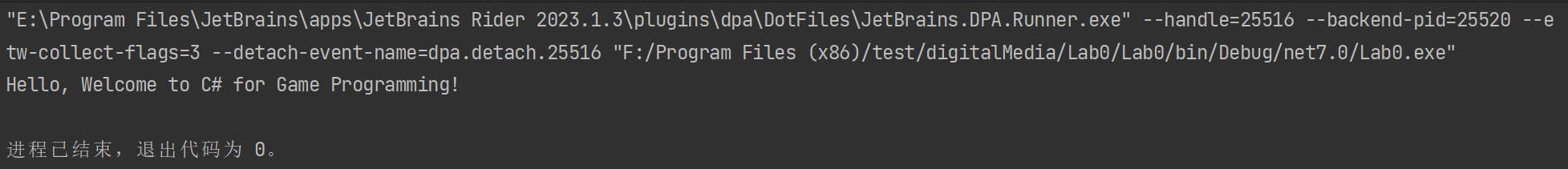
1.1

The console output the string “Hello, Welcome to C# Game Programming” with exit code 0. Because I’m using rider, the IDE called a console itself. So it won’t disappear soon. However I can add a line of code such as Console.ReadKey(); at the end of the Main method. This will wait for a key press before closing the console window.

1.2

Source Code:

using System;  
  
namespace Lab1  
{  
 class Person  
 {  
 private string \_name;  
 private int \_age;  
 private double \_height;  
 private double \_weight;  
   
 public string GetName() { return \_name; }  
 public int GetAge() { return \_age; }  
 public double GetHeight() { return \_height; }  
 public double GetWeight() { return \_weight; }  
   
 public void SetName(string n) { \_name = n; }  
 public void SetAge(int a) { \_age = a; }  
 public void SetHeight(double h) { \_height = h; }  
 public void SetWeight(double w) { \_weight = w; }  
 }  
   
 class Program  
 {  
 static void Main(string[ ] args)   
 {   
 Person David = new Person();   
 Person Alice = new Person();   
  
*// Initialize David* David.SetAge(21);   
 David.SetName("David");   
 David.SetWeight(185.4);   
 David.SetHeight(72.3);   
  
*// Initialize Alice* Alice.SetAge( 18);   
 Alice.SetName("Alice");   
 Alice.SetWeight(125.7);  
 Alice.SetHeight(67.1);  
  
*// print some values* Console.WriteLine("David’s age = {0}; David’s weight = {1}",David.GetAge(), David.GetWeight());   
 Console.WriteLine("Alice’s age = {0}; Alice’s weight = {1}", Alice.GetAge(), Alice.GetWeight());   
 }   
  
 }  
}

1.3

Source Code

using System;  
  
namespace Lab1  
{  
 class Person  
 {  
 private string \_name;  
 private int \_age;  
 private double \_height;  
 private double \_weight;  
   
 public string Name  
 {  
 get { return \_name; }  
 set { \_name = value; }  
 }  
   
 public int Age  
 {  
 get { return \_age; }  
 set { \_age = value; }  
 }  
   
 public double Height  
 {  
 get { return \_height; }  
 set { \_height = value; }  
 }  
   
 public double Weight  
 {  
 get { return \_weight; }  
 set { \_weight = value; }  
 }  
 }  
   
 class Program  
 {  
 static void Main(string[ ] args)   
 {   
 Person David = new Person();   
 Person Alice = new Person();   
  
*// Initialize David* David.Age = 21;  
 David.Name = "David";  
 David.Weight = 185.4;  
 David.Height = 72.3;  
  
*// Initialize Alice* Alice.Age = 18;  
 Alice.Name = "Alice";  
 Alice.Weight = 125.7;  
 Alice.Height = 67.1;  
  
*// print some values* Console.WriteLine("David’s age = {0}; David’s weight = {1}",David.Age, David.Weight);   
 Console.WriteLine("Alice’s age = {0}; Alice’s weight = {1}", Alice.Age, Alice.Weight);   
 }   
  
 }  
}

1.4

(1) The purpose of a constructor in C# is to initialize an object’s members when the object is created. They can be used to set default values for data members, validate input parameters, or perform any other initialization tasks that are necessary for the object to function properly.

(2) A default constructor is a constructor that takes no parameters. It is automatically provided by the C# compiler if no other constructor is defined for a class.

(3) Programmer don’t have to provide a default constructor for the class. If not provide any constructors, the C# compiler will automatically generate a default constructor for the class. This default constructor will initialize all of the object’s data members to their default values.

1.5

Source Code

using System;  
  
namespace Lab1  
{  
 class Person  
 {  
 private string \_name;  
 private int \_age;  
 private double \_height;  
 private double \_weight;  
 public string Name  
 {  
 get { return \_name; }  
 set { \_name = value; }  
 }  
 public int Age  
 {  
 get { return \_age; }  
 set { \_age = value; }  
 }  
 public double Height  
 {  
 get { return \_height; }  
 set { \_height = value; }  
 }  
 public double Weight  
 {  
 get { return \_weight; }  
 set { \_weight = value; }  
 }  
 public Person()  
 {  
 \_name = "No name";  
 \_age = 0;  
 \_height = 0;  
 \_weight = 0;  
 }  
 public Person(string name, int age, double height, double weight)  
 {  
 \_name = name;  
 \_age = age;  
 \_height = height;  
 \_weight = weight;  
 }  
   
 }  
   
 class Program  
 {  
 static void Main(string[ ] args)   
 {   
 Person David = new Person("David",21, 72.3, 185.4);  
 Person Alice = new Person("Alice",18, 67.1, 125.7);  
*// print some values* Console.WriteLine("David’s age = {0}; David’s weight = {1}",David.Age, David.Weight);   
 Console.WriteLine("Alice’s age = {0}; Alice’s weight = {1}", Alice.Age, Alice.Weight);   
 }   
  
 }  
}

1.6

(1) To provide a common interface and base functionality for a set of related classes

(2) There will get a compile-time error. Abstract classes are meant to be used as base classes and cannot be instantiated on their own.

(3) The Area property is defined as abstract because the calculation of the area depends on the specific shape being represented. The Area is read-only because it doesn’t make sense to set the area of a shape directly, instead, the area should be calculated based on the shape.

(4) The override keyword is used to indicate that the ToString method in the Shape class is intended to override the ToString method inherited from the base Object class. If without using the override keyword, there will get a compile-time warning and when calling the ToString method on an instance of a derived class, the version of the method defined in the base Object class will be called instead of the version defined in the Shape class.

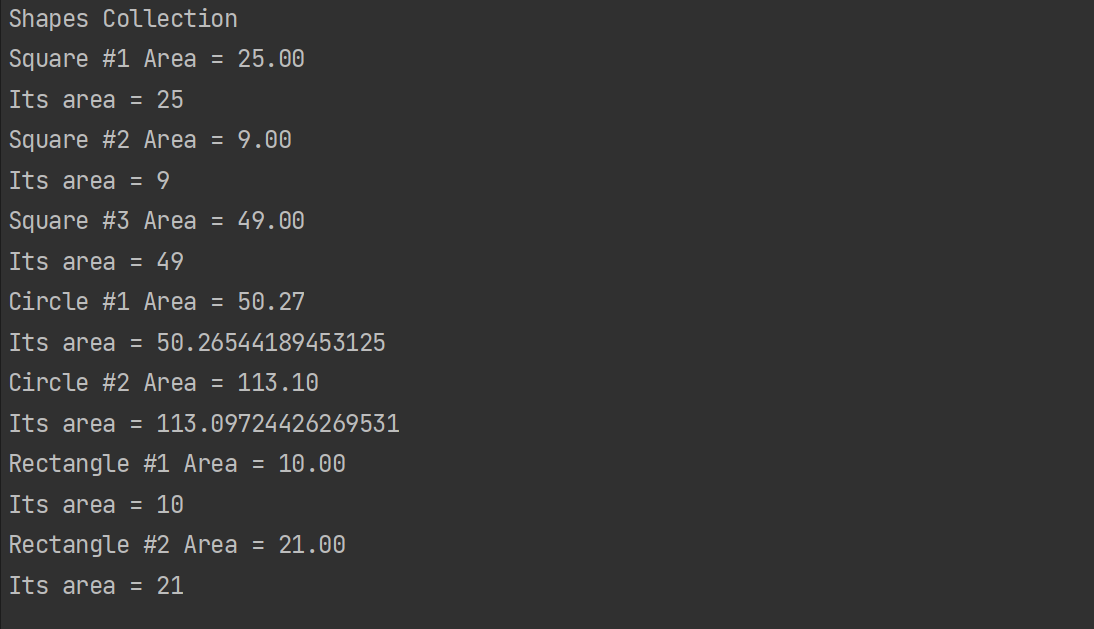
1.7



(1)

Source Code

using System;  
  
namespace Lab1  
{  
 public abstract class Shape  
 {  
 private string myId;  
  
 public Shape(string s)  
 {  
 Id = s; *// calling the set accessor of the Id property* }  
  
 public string Id  
 {  
 get { return myId; }  
  
 set { myId = value; }  
 }  
  
 *// Area is a read-only property - only a get accessor is needed:* public abstract double Area { get; }  
  
 public override string ToString()  
 {  
 return Id + " Area = " + string.Format("{0:F2}", Area);  
 }  
 }  
  
 public class Square : Shape  
 {  
 private int mySide;  
  
 public Square(int side, string id) : base(id)  
 {  
 mySide = side;  
 }  
  
 public override double Area  
 {  
 get  
 {  
 *// Given the side, return the area of a square:* return mySide \* mySide;  
 }  
 }  
 }  
  
 public class Circle : Shape  
 {  
 private int myRadius;  
  
 public Circle(int radius, string id) : base(id)  
 {  
 myRadius = radius;  
 }  
  
 public override double Area  
 {  
 get  
 {  
 *// Given the side, return the area of a square:* return myRadius \* myRadius \* 3.14159f;  
 }  
 }  
 }  
  
 public class Rectangle : Shape  
 {  
 private int myWidth;  
 private int myHeight;  
  
 public Rectangle(int width, int height, string id) : base(id)  
 {  
 myWidth = width;  
 myHeight = height;  
 }  
  
 public override double Area  
 {  
 get  
 {  
 return myWidth \* myHeight \* 0.5f;  
 }  
 }  
 }  
  
 public class Program  
 {  
 public static void Main()  
 {  
 Shape[] shapeArray =  
 {  
 new Square(5, "Square #1"),  
 new Square(3, "Square #2"),  
 new Square(7, "Square #3"),  
 new Circle(4, "Circle #1"),  
 new Circle(6, "Circle #2"),  
 new Rectangle(4, 5, "Rectangle #1"),  
 new Rectangle(6, 7, "Rectangle #2")  
 };  
  
 System.Console.WriteLine("Shapes Collection");  
 foreach (Shape s in shapeArray)  
 {  
 System.Console.WriteLine(s);  
 System.Console.WriteLine("Its area = {0}", s.Area);  
 }  
 }  
 }  
}



(2) it is called polymorphism