# Unit 2 - Exercises

The following exercises will requires additional files. You can download the files [here](https://thethirstycoder.gitlab.io/adev-2005-learn/unit_2/downloads/adev-2005_unit_2_exercise_files.zip).

Create a new Visual Studio solution called ADEV2005Unit2 containing a Console Application project called Unit2Exercises.

## Properties

### Setup

For this exercise, files from a previous exercise are required.

1. In File Explorer, copy the PlayingCard.cs, CardRank.cs, and CardSuit.cs files from the Unit1Exercises project directory.
2. Paste the copied files into the Unit2Exercises project directory.
3. With the Unit2Exercises project loaded in Visual Studio, right-click on the Unit2Exericses project in Solution Explorer and choose Add > Existing item… from the context menu.
4. Select the three files and click the “Add” button.

**EX1**. Update the PlayingCard class to use properties rather than the accessor and mutator methods.

1. Remove class members:
2. Add and implement class properties:
3. In the Program class, demonstrate the use of each property.

## Inheritance and Abstract Classes

**EX2**. Add a second project to your ADEV2005Unit2 Visual Studio solution.

1. Right-click on the “ADEV2005Unit2” node in the Solution Explorer panel. Choose Add > New Project… from the context menu.
2. Choose “Class Library” as the project type from the middle panel of the Add New Project dialog window.
3. Enter ADEV2005API in the project Name text field.
4. Click the “Ok” button.

This will create a new project within the existing solution. Your exercises solution now has two projects.

**EX3**. Add an existing file to a project.

1. In the Windows File Explorer application, copy the provided Employee.cs file to the “ADEV2005API” project directory.
2. Right-click on the “ADEV2005API” project in the Solution Explorer in Visual Studio and choose Add > Existing item… from the context menu.
3. Locate the Employee.cs file and select it.
4. Click the “Add” button.
5. Review the code within the Employee.cs file.

**EX4**. Create a class called HourlyEmployee which derives from the Employee class.

1. Add a new code file called HourlyEmployee.cs to the ADEV2005API project.
2. Code the class in the namespace Ca.Rrc.Adev2005.
3. Declare the HourlyEmployee class. The HourlyEmployee class will derive from the Employee class.

**EX5**. The HourlyEmployee class currently contains no code. But it does derive from the Employee class. Which means HourlyEmployee does have class members. Identify them:

Attributes:

* \_**\_\_\_\_\_\_\_\_\_\_**\_
* \_**\_\_\_\_\_\_\_\_\_\_**\_

Behaviors

* \_**\_\_\_\_\_\_\_\_\_\_**\_
* \_**\_\_\_\_\_\_\_\_\_\_**\_
* \_**\_\_\_\_\_\_\_\_\_\_**\_
* \_**\_\_\_\_\_\_\_\_\_\_**\_

**EX6**. In the Program class, construct two employee objects.

1. Construct an instance of the Employee class.
2. Construct an instance of the HourlyEmployee class.

### Questions

1. How are you able to construct instances of these classes when the classes contain no constructor methods?

**EX7**. Add a constructor to the Employee class.

+ Employee(employeeID : string, name : string) - Constructs an instance of the Employee class with the specified employee id and name.

### Questions

Before answering the following questions, build the project.

1. Building the project at this point will cause a syntax error. Note the description of the error. What is causing the error?

**EX8**. Add a constructor to the HourlyEmployee class.

+ HourlyEmployee(employeeID : string, name : string) - Constructs an instance of the HourlyEmployee class with the specified employee id and name.

Implement this constructor such that it invokes the constructor of its base class. Completing this step will remove the error generated in the previous exercise.

Update the Program such that the employees are constructed using the new constructor methods.

Employee:

* Name: Dan
* ID: DD333

HourlyEmployee:

* Name: James
* ID: JB007

**EX9**. Invoking inherited behaviours.

1. Update the Program to invoke the Terminated behaviour of each Employee object.
2. Run the application and note the results.

**EX10**. Override the Terminated method from the Employee class in the HourlyEmployee class.

1. Add a Terminated method to the HourlyEmployee class to override the Terminated method from the base class Employee. The implementation of the method will return the literal string “HourlyEmployee has be terminated.”
2. Build the project.

### Questions

1. After building the project, a warning is generated. Why is this warning happening?

**EX11**. Update the declaration of the Terminated method in the Employee class to prevent the warning generated in the previous exercise.

1. Modify the declaration of the Terminated method in the Employee class to allow the override to happen.
2. Build the project.
3. Run the application and note the results.

### Questions

1. As it pertains to method overriding, what is the coding difference between C# and Java?

**EX12**. Declare the Employee class as an abstract class.

1. Declare the Employee class as an abstract class.
2. Build the project.

### Questions

1. A syntax error is generated in the program. Note the description of the error. Why is this error generated?

**EX13**. Update the program to eliminate the error.

1. Make the appropriate change to eliminate the error.
2. Build the project.

### Questions

1. Why can we code a constructor in an abstract class if we cannot construct instances of it?

**EX14**. Add an abstract method to the Employee class.

1. Declare the GetPay method with the Employee class. GetPay() : decimal - Returns the Employee’s pay.
2. Build the project.

### Questions

1. A syntax error is generated. Note the description of the error. Why is this error generated?

**EX15**. Update the HourlyEmployee class to implement the abstract method from the Employee class. Steps

1. Declare two fields within the HourlyEmployee class.
   * hoursWorked : decimal
   * rateOfPay : decimal
2. Update the HourlyEmployee constructor, to set the state of these fields. Both are set to zero.
3. Declare (and override) the GetPay method within the HourlyEmployee class. The implementation is:
4. decimal overtimeHoursWorked = Math.Max(\_hoursWorked – 40, 0);
5. decimal regularHoursWorked = \_hoursWorked – overtimeHoursWorked;
6. return (\_rateOfPay \* regularHoursWorked) + (\_rateOfPay \* 1.5m \* overtimeHoursWorked);
7. Build the project.
8. Update the program to test the GetPay method. Note: the GetPay method has more than one outcome. Ensure you are testing all outcomes.

**EX16**. Create a polymorphic reference.

HourlyEmployee Class

1. Add a property called HoursWorked that gets and sets the hours the HourlyEmployee worked.
2. Build the project.

Program Class

1. Update the program by declaring a variable of Employee type. The name of the variable is myEmployee.
2. Assign the myEmployee variable to a new instance of HourlyEmployee.
3. Using the myEmployee variable, print the number of hours the employee worked.
4. Run the program.

### Questions

1. A runtime error is generated. Note the description of the error. Why is this error generated?

**EX17**. Cast an object.

The syntax for casting in C# is the same as Java.

1. Update the program to cast the myEmployee variable to a HourlyEmployee before printing the hours. You can create a new reference variable, or preform the cast and get the HoursWorked in a single step.

## Exceptions

**EX18**. Throwing exceptions.

Exceptions are generated by throwing an instance of an Exception.

1. Update the implementation of the constructor in the Employee class.
   * If the name or id contains no characters (do a Trim() on these fields before checking), throw an ArgumentException(message : string, paramName : string).
2. The message will read “The argument cannot be blank.”
3. Update the implementation of the constructor in the HourlyEmployee class.
   * If the hours worked or rate of pay is less than zero, throw an ArgumentOutOfRangeException(paramName : string, message : string). The message will read “The argument cannot be a negative value.”

### Questions

1. What does throwing exceptions accomplish from the perspective of the developer who coded the implementation?

**EX19**. Generating an exception.

As a developer, generating an exception is typically something that is avoided. But when testing our code, we want to generate the exception to ensure it is happening when it should.

1. In the Main method of the Program class, construct an instance of the HourlyEmployee class. Use any value you like for the arguments. For each test, change the value of one of the arguments to cause an Exception.
2. Print the amount paid to the employee.
3. Run the application. The application will abend and will activate Visual Studio, highlighting the line of code where the exception was generated.
4. Read the exception message in the window that appears. Does it match the message you coded?
5. Click the Stop Debugging button in the toolbar to stop executing the application.
6. Repeat this process for each argument of the method.

### Questions

1. What does producing an exception do for the developer who used a method that generated an exception?

**EX20**. Handling exceptions.

When using methods that contain code to throw an Exception, you may need to handle the exception to control the flow of your code. This is accomplished using a try…catch construct.

1. Try to construct and print the pay of the HourlyEmployee.
2. Catch the Exception that was generated. Print “An error occurred calculating the employee’s pay.” to the console. Also print:
   * The parameter name that caused the exception.
   * The exception message.
   * The stack trace.

### Questions

1. Why are both the construction and print statements coded within the Try block?
2. What happens if the print statement is after the Catch block?

## Auto-implemented Properties

**EX21**. Coding an auto-implemented property.

1. Add an auto-implemented property to the Employee class called Position. The type is String. The property will provide public access to get and set the Employee’s position.

### Questions

1. How does an instance of Employee store state for the Position property without a field declared?
2. Update the constructor of the Employee class to include a parameter for Position.
3. Throw an ArgumentException when the value of the argument is blank.
4. In the Main method of the Program class, print the position of the instance of HourlyEmployee created in a previous skill-check.
5. Assume you do not know how the Employee class is coded. By the public interface of the Employee class, can you tell that the Position property is auto-implemented? Explain why or why not.

## Static Classes

**EX22**. Code a static class called Conversion.

1. Add a new code file to the “ADEV2005API” project. The name of the file is Conversion.cs.
2. Declare a static class called Conversion in the Ca.Rrc.Adev2005 namespace.
3. Declare the following fields in the class:
   * course : string = ADEV-2005 (constant)
4. Declare and implement two static methods:
   * A method that converts a measurement in Feet to Meters. The formula for the implementation is: Meters = Feet x 0.305
   * A method that converts a measurement in Miles to Kilometers. The formula for the implementation is: Kilometers = Miles x 1.61

**EX23**. Test the static class in the program.

1. Print the value of the field within the Conversion class.
2. Invoke each method in the Conversion class and print the results to the screen. Use your own test data. No user input is required.

### Questions

1. What is a static class used for?
2. How do C# static classes differ from “static” classes in Java?
3. Can you construct instances of a static class (yes/no)? If yes, explain how that is done. Use code to support your answer.