Abstract— One of the main and effective tools when it comes to organizing academic activities for students in computer science disciplines is creating a problem situation. They require an “outside the box” mindset, as well as systematization of knowledge and skill. At the same time, the developed tasks the students are presented with at the current learning level may require neglecting some nonessential factors of the problem in the absence of knowledge. The difficulty level of tasks must match the preparation level of the students. This paper offers a methodological approach for teaching basic concepts of exception handling in C# for first year students, specialty Software technologies and design.

Keywords— learning programming, teaching computer science, exception handling, education

I. INTRODUCTION

The educational model in programming is a more defined and in-depth continuation of the general theory of education, which is based on a specific educational system. At the core of the educational model is the problem-based learning method, therefore, the learning process consists of a series of problem situations. Programming education is inherently problem-based and is carried out through tasks.

II. PROBLEMS AS A LEARNING TOOL

One of the main and effective tools when it comes to organizing academic activities for students in computer science disciplines is creating a problem situation. They require an “outside the box” mindset, as well as systematization of knowledge and skill. At the same time, the developed tasks the students are presented with at the current learning level may require neglecting some nonessential factors of the problem in the absence of knowledge. The difficulty level of tasks must match the preparation level of the students. A problem must be defined in such a way that it requires basic knowledge of the respective learning level it’s meant for. When formulating the conditions of the task there must be a problem that cannot be solved by any known means [4].

In the study of informatics it is necessary to solve a problem that arises outside of the programming subject area – be it science, technology, economy or any other practical field, as well as formulating respective tasks. Then one must solve these problems by means of information modelling, i.e. create a program and conduct an experiment. Problems in programming differ from problems in mathematics. The programmer creates an information model within the framework of computer resources – there are restrictions on the format and scope of the data, the time it would take to come up with a solution, the model’s level of adequacy and the real object of the study. Examples and tasks are needed to consolidate and apply knowledge, related to the principal concepts introduced.

These tasks also have several specific functions: to help in developing an algorithmic thinking style, mastering the basics of modelling, formulation in detecting the connection between different forms of the same concept, phenomenon or process. In the process of solving a problem, students consolidate their knowledge and also discover new characteristics of previously studied elements through activity on their part. One of the goals of programming education is for its students to put theory to practice by solving practical problems. This goal can be reached through a system of tasks. A system of tasks consists of methodical series of tasks, ensuring planned educational results [17]. Every task in such a system contains specific information related to the learning material and has its own place and purpose, as tasks are arranged in a progressively difficult manner. The goal of problems in these systems is to ensure that all stages of learning are covered: introduction, consolidation, application. Consolidation tasks improve the understanding of concepts and their essential characteristics as well as their use. Emphasis on the question „Why does it need to be done like this?” Knowledge is considered utilized when students are able to apply it freely.
This is why the application task group must be such as to create a basic idea of the application limits of the respective element in addition to its typical use. The development of skills in order to apply knowledge has different stages – recognition and reproduction, analysing, evaluation, creation. The purpose of some tasks within a system is to ease learning and skill application on different levels, while others are meant for individual and group work and create rational feedback conditions.

III. EXCEPTION HANDLING

Errors can occur at almost any statement. They can occur for almost any reason. Exception handling separates error-related logic. It simplifies control flow. Exceptions in C# provide a structured, uniform, and type-safe way of handling both system-level and application-level error conditions. Most object-oriented programming languages support a feature called exception mechanism.

A. Exception mechanism in the Object-Oriented Programming

Since the object-oriented languages work with objects that should come as close as they can to real life the need for error management occurs. When the client code tries to use the functionality of an object in an unsupported way the object should be able to deny the service. In other cases when the object relies on a certain service but reaches a situation where this service is denied it again should be able to notify its client code of the unexpected situation. In both situations the object should be able to explain the situation to its client code with an informative error. Exceptions and exception handling are actually important tools in the programmer's repertoire [11]. Many things can cause a program to stop working. These include:

- **Programmer error**, such as failing to check the denominator for 0 before performing a division or trying to access property values of a null object;
- **Interaction with system resources**, for example the operating system doesn't have enough memory to acquire a needed object for your program or the printer is tied up when you try to print; loss of network connection etc.;
- **User error**, such as trying to save to a read-only CD or looking for a file in a path that doesn't exist.

It is not always easy to determine where a problem first appears. And it might take a while for the problem to be detected. An exception is an object which contains information about the problem.

**What is an exception?** An indication of an unusual situation, such as a failure during execution: division by zero, memory access violation, array out of bounds, etc.

We can create our own exceptions and throw them (also termed raising an exception) when problematic situations arise in our objects (for example we could raise an exception if we try to divide by zero). Most often, objects from the .NET framework will raise exceptions and we should catch these (i.e. decide what to do), dealing with exceptions usually entails closing the program gracefully and providing a message about the problem.

B. Exception handling in .NET framework

Exception handling is an in built mechanism in .NET framework to detect and handle runtime errors. The .NET framework contains lots of standard exceptions.

The exceptions are caused or "throw" (throw an exception) from programming code that needs to signal to the executing program for error or unusual situation. In the C# language, you can throw exceptions with a throw statement. They can also be thrown automatically by the runtime. This usually occurs because of the values of the variables in your code.

C# provides three keywords try, catch and finally to do exception handling. The try encloses the statements that might throw an exception, whereas catch handles an exception if one exists. The finally can be used for doing any clean-up process.

If any exception occurs inside the try block, the control transfers to the appropriate catch block and later to the finally block. But in C#, both catch and finally blocks are optional. The try block can exist either with one or more catch blocks or a finally block or with both catch and finally blocks. If there is no exception occurred inside the try block, the control directly transfers to finally block. Therefore the statements inside the finally block is executed always. Note that it is an error to transfer control out of a finally block by using break, continue, return or goto.

C. Exception Classes in C#

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the System.Exception class. Some of the exception classes derived from the System. Exception class are:

- **System.ApplicationException class** - supports exceptions generated by application programs. So the exceptions defined by the programmers should derive from this class.
- **System.SystemException class** - the base class for all predefined system exception.

The following table provides some of the commonly used exceptions defined within the System Namespace – System.Exception class [21]:
### TABLE 1

<table>
<thead>
<tr>
<th>Exception Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOException</td>
<td>Handles I/O errors.</td>
</tr>
<tr>
<td>IndexOutOfRangeException</td>
<td>Array index is out of bounds.</td>
</tr>
<tr>
<td>ArrayTypeMismatchException</td>
<td>Type of value being stored is incompatible with the type of the array.</td>
</tr>
<tr>
<td>NullReferenceException</td>
<td>Handles errors generated from referencing a null object.</td>
</tr>
<tr>
<td>DivideByZeroException</td>
<td>Division by zero attempted.</td>
</tr>
<tr>
<td>InvalidCastException</td>
<td>A runtime cast is invalid.</td>
</tr>
<tr>
<td>OverflowException</td>
<td>An arithmetic overflow occurred.</td>
</tr>
<tr>
<td>OutOfMemoryException</td>
<td>A call to new fail because insufficient free memory exists.</td>
</tr>
<tr>
<td>StackOverflowException</td>
<td>The stack was overrun.</td>
</tr>
</tbody>
</table>

We will present examples of tasks for acquiring skills for exception handling in C#. The students are first academic year, specialty “Software technologies and design”, and they have basic knowledge of Programming. Students study successively the methods and means of the structured programming – basic control structures, abstraction of data (types and data structures and their presentation), compilation and analysis of basic algorithms.

IV. SOME SAMPLE TASKS FOR ACQUIRING SKILLS FOR EXCEPTION HANDLING IN C#

A. Tasks, that use try and catch

At the core of exception handling are try and catch. These keywords work together, and you can’t have a catch without a try. This is a simple example that illustrates how to watch for and catch an exception:

```csharp
class Program
{
    static void Main()
    {
        try
        {
            int value = 10 / int.Parse("0");
            Console.WriteLine(value);
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine(ex.Message); } 
    }
}
```

When an exception is thrown, it is caught by its corresponding catch clause, which then processes the exception. In the general form, there can be more than one catch clause associated with a try. The type of the exception determines which catch is executed. That is, if the exception type specified by a catch matches that of the exception, then the block of code associated with that catch clause is executed. When the above code is compiled and executed, it produces following result:

```
Attempted to divide by zero.
```

The next example shows frequently detectable error, that attempt to index an array beyond its boundaries.

```csharp
class ExcepTest
{
    static void Main()
    {
        int[] nums = new int[10];
        try
        {
            Console.WriteLine("Before except. is generated.");
            // Generate an index out-of-bounds exception.
            nums[10] = 100;
            Console.WriteLine("this won't be displayed");
        }
        catch (IndexOutOfRangeException)
        { // catch the exception
            Console.WriteLine("Index out-of-bounds!");
        }
        Console.WriteLine("After catch block.");
        Console.ReadKey(); } 
}
```

B. Tasks, that use Multiple Catch Blocks

You can associate more than one catch clause with a try. In fact, it is common to do so. However, each catch must catch a different type of exception. For example below, the program shown here catches both array-boundary and divide-by-zero errors:

```csharp
class Program
{
    static void Main()
    {
        try
        {
            int value = 10 / int.Parse("0");
            Console.WriteLine(value);
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine(ex.Message); }
        catch (IndexOutOfRangeException)
        { // catch the exception
            Console.WriteLine("Index out-of-bounds!");
        }
    }
}
```
As the output confirms, each catch clause responds only to its own type of exception. Catch clauses are checked in the order in which they occur in a program, but only a matching clause is executed, all others are ignored. Often you will want to catch all exceptions, no matter the type.

The hierarchical nature of the exceptions allows at once catching and handling exceptions of entire groups. When we use the catch, we offset only the type of exception, the entire hierarchy of the types of exceptions declared heirs of our type. To do this, use a catch clause that specifies no exception type at all.

C. Tasks, that use finally block

Sometimes, you will want to define a block of code that will execute when a try/catch block is left. For example, an exception might cause an error that terminates the current method, causing its premature return.

```csharp
class ExcMulti
{
    static void Main()
    {
        // Here, numer is longer than denom.
        int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };  
        int[] denom = { 2, 0, 4, 4, 0, 8 }; 
        for (int i = 0; i < numer.Length; i++)
        {
            try
            {
                Console.WriteLine(numer[i] + " / " + denom[i] + " is " + numer[i] / denom[i]);
            } catch (DivideByZeroException)
            {
                Console.WriteLine("Can't divide by Zero!");
            } catch (IndexOutOfRangeException)
            {
                Console.WriteLine("No matching element found.");
            }
            finally
            {
                Console.WriteLine("Result: {0}", result);
            }
        }
    }
}
```

However, that method may have opened a file or a network connection that needs to be closed. Such types of circumstances are common in programming, and C# provides a convenient way to handle them: finally. To specify a block of code to execute when a try/catch block is exited, include a finally block at the end of a try/catch sequence. The general form of a try/catch that includes finally is shown above.

The finally block will be executed whenever execution leaves a try/catch block, no matter what conditions cause it. Whether the try block ends normally or because of an exception, the last code executed is that defined by finally. The finally block is also executed. Here is an example of finally:

```csharp
public class DivNumbers
{
    int result;
    DivNumbers()
    {
        result = 0;
    } public void division(int num1, int num2)
    {
        try
        {
            result = num1 / num2;
        } catch (DivideByZeroException e)
        {
            Console.WriteLine("Exception caught: {0}", e);
        } catch (ArithmeticException e)
        {
            Console.WriteLine("Arithmetic error: {0}", e);
        } finally
        {
            Console.WriteLine("Result: {0}", result);
        }
    }
    static void Main(string[] args)
    {
        DivNumbers d = new DivNumbers();
        d.division(25, 0);
        Console.ReadKey();
    }
}
```

In this case, it doesn’t matter what exception is is thrown or not thrown the file. Always will produce following result: Result: 0.
So far, we looked at the use of try-finally release only one resource, but sometimes it may need to release more than one resource. It is good practice to release resources in reverse order to that of their allocation.

D. Tasks, that use IDisposable and using Statement

The primary use of IDisposable interface is to release resources. In the .NET such resources are graphic elements (window handles), files, streams and etc. The Important method in the interface IDisposable is Dispose(). The main, which you must to know about it, is that he frees resources of class that implements it.

Statement using provides a convenient syntax that ensures the correct use of IDisposable objects. The C# "using" statement results in a call to Dispose(). This is the same as Close(), which may throw exceptions when a network error occurs. Because the call to Dispose() happens implicitly at the closing brace of the "using" block, this source of exceptions is likely to go unnoticed both by people writing the code and reading the code. This represents a potential source of application errors.

So the closing of a stream can be done in the following way:

```csharp
StreamReader reader = new StreamReader(fileName);
try {
    // Use the reader here
}
finally {
    if (reader != null) {
        reader.Dispose();
    }
}
```

The last example can be written with statement using in C# language in the following way:

```csharp
using (StreamReader reader = new StreamReader(fileName)) {
    // Use the reader here
}
```

This version seems much shorter and clearer. It is necessary neither to have try-finally, nor cry any explicit methods for the release of resources. The compiler takes care to put an automatic try-finally block, which when exiting the using block or reaching its closing brace, to call the method Dispose() to release the resources used in the block.

The following example is for recording and reading of data to or from a text file with statement using.

```csharp
string[] names = new string[] {"Anna", "Nina", "Petar", "Ivan"};
using (StreamWriter sw = new StreamWriter("names.txt"))
{
    foreach (string s in names)
    {
        sw.WriteLine(s);
    }
    // Read and show each line from the file.
    string line = "";
    using (StreamReader sr = new StreamReader("names.txt"))
    {
        while ((line = sr.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
```

Statements using can be put in one another.

Multiple instances of a type can be declared in a using statement, as shown in the following example:

```csharp
using (Font font3 = new Font("Arial", 10.0f), font4 = new Font("Arial", 10.0f))
{
    // Use font3 and font4.
}
```

It is important to note that the construction using no treatment of exceptions. Its only role is to free resources independently whether exceptions were thrown or not and what the possible exceptions were thrown. You should use statement using when working with all classes that implement IDisposable.

E. Tasks, that use throwing an Exception

The preceding examples have been catching exceptions generated automatically by the runtime system. However, it is possible to manually throw an exception by using the throw statement. Its general form is shown here:

```csharp
throw exceptOb;
```

For example the following statement throw a DivideByZeroException explicitly:

```csharp
class ThrowException {
    static void Main() {
        try {
            Console.WriteLine("Before throw.");
            throw new DivideByZeroException();
        }
        catch (DivideByZeroException) {
            Console.WriteLine("Exception caught.");
        }
        Console.WriteLine("After try/catch statement.");
    }
}
```
F. Tasks with rethrowing an Exception

An exception caught by one catch clause can be
rethrown so that it can be caught by an outer catch. The
most likely reason for rethrowing an exception is to allow
multiple handlers access to the exception. For example,
perhaps one exception handler manages one aspect of an
exception, and a second handler copes with another aspect.
To rethrow an exception, you simply specify throw,
without specifying an exception.

The following program illustrates rethrowing an exception:

```csharp
class RetrowException
{
    public static void Main()
    {
        int[] table = new int[] { 10, 11, 12, 13, 14, 15 };
        int idx = 6;
        M(table, idx);
    }
    public static void M (int[] table, int idx)
    {
        try
        {
            Console.WriteLine("Accessing element {0}: {1}\n", idx, table[idx]);
        }
        catch (NullReferenceException)
        {
            Console.WriteLine("A null reference exception");
            throw; // rethrowing the exception
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Divide by zero");
            throw; // rethrowing the exception
        }
    }
}
```

In console applications errors are usually printed in the
output although this might not be the most user-friendly
way to notify the user for problems.

In Web applications, errors are frequently shown in the
beginning or at the bottom of the page or near the user
interface field related to the error. An example – task for
reading from a text file and its visualization in a text box.
Unless you specify the correct file name or not found, an
exception occurs.

```csharp
private void button1_Click(object sender, EventArgs e)
{
    string file_name = "test.txt";
    if (File.Exists(file_name) == true)
    {
        StreamReader objReader = new
        StreamReader(file_name);
        textBoxOpen.Text = objReader.ReadToEnd();
        objReader.Close();
    }
    else
        MessageBox.Show("No such file "+ file_name,
        "MessageBox", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

As you can see, there is no single 'right' way to handle and
visualize exceptions as it depends on the type of the
application and its intended audience.
V. CONCLUSION

Tasks have a very specific role in the learning of informatics and information technologies. In particular, they can be educational tools within themselves. In their studies, a significant number of authors [1, 7, 8, 9, 13] view systems of educational tasks as a powerful tool for managing the preparation of informatics and information technologies students throughout different stages of the educational process.

The new and fast-changing content of Informatics teaching requires the development of methodologies which could provide not only reproduction of large volume of knowledge, but most of all methodologies, which could form and develop competences that would allow students to master the knowledge actively and to create skills for acquiring new knowledge independently and for comprehending it critically. In the process of solving a problem, students consolidate their knowledge and also discover new characteristics of previously studied elements through activity on their part. Results from the study show, that the interest of students is stimulated, their attention and motivation are promoted for the implementation of accumulated procedural knowledge for use in new context.

REFERENCES