**Тема: «Структурная обработка исключений в .NET, проектирование собственных типов исключений»**

**Задание 1: Создание пользовательского исключения**

**Общие требования:**

В каждой задаче необходимо:

1. Создать **класс пользовательского исключения**, наследуемый от Exception, который содержит:
   * Конструкторы (по умолчанию и с сообщением).
   * Возможность передачи InnerException.
2. Создать **класс с методом**, в котором выбрасывается пользовательское исключение.
3. В Main-методе **продемонстрировать обработку исключения** с помощью try-catch.

**Задачи:**

1. **Проверка возраста пользователя**
   * Создать исключение InvalidAgeException, выбрасываемое при вводе возраста меньше 18 лет.
   * Создать класс UserAgeValidator с методом ValidateAge(int age), который проверяет введенный возраст и выбрасывает исключение при некорректном значении.
2. **Валидация банковского счета**
   * Создать исключение InvalidAccountNumberException, если номер банковского счета состоит не из 10 цифр.
   * Класс BankAccount должен содержать метод ValidateAccount(string accountNumber), проверяющий корректность номера счета.
3. **Проверка баланса банковского счета**
   * Создать исключение InsufficientFundsException, если на счете недостаточно средств для снятия.
   * Класс BankAccount должен иметь метод Withdraw(decimal amount), который выполняет проверку баланса перед операцией снятия.
4. **Проверка пароля**
   * Создать исключение WeakPasswordException, если пароль короче 8 символов.
   * Класс PasswordValidator должен содержать метод ValidatePassword(string password), проверяющий длину пароля.
5. **Ограничение скорости автомобиля**
   * Создать исключение SpeedLimitExceededException, если скорость автомобиля превышает 120 км/ч.
   * Класс Car должен иметь метод SetSpeed(int speed), проверяющий допустимый предел скорости.
6. **Проверка температуры в термостате**
   * Создать исключение OverheatException, если температура превышает 100°C.
   * Класс Thermostat должен содержать метод CheckTemperature(int temperature), контролирующий допустимый диапазон значений.
7. **Запрет на удаление главного администратора**
   * Создать исключение AdminDeletionException, если происходит попытка удалить пользователя с ролью "Admin".
   * Класс UserManager должен иметь метод DeleteUser(string role), выполняющий проверку перед удалением.
8. **Контроль количества товаров на складе**
   * Создать исключение OutOfStockException, если количество товара на складе меньше 1.
   * Класс Warehouse должен иметь метод CheckStock(int quantity), проверяющий наличие товара.
9. **Ошибка в формате электронной почты**
   * Создать исключение InvalidEmailFormatException, если введенный email не содержит @ или ..
   * Класс EmailValidator должен иметь метод ValidateEmail(string email), проверяющий формат email-адреса.
10. **Проверка уровня заряда батареи**
    * Создать исключение LowBatteryException, если заряд батареи ниже 5%.
    * Класс BatteryMonitor должен иметь метод CheckBatteryLevel(int level), проверяющий уровень заряда.
11. **Ограничение на количество попыток входа в систему**
    * Создать исключение TooManyLoginAttemptsException, если количество попыток превышает 3.
    * Класс LoginManager должен иметь метод AttemptLogin(int attempts), контролирующий число попыток входа.
12. **Ошибка при расчете среднего балла студента**
    * Создать исключение InvalidGradeException, если введена оценка вне диапазона 0–100.
    * Класс StudentGrade должен содержать метод CalculateAverage(List<int> grades), проверяющий корректность оценок.
13. **Проверка формата телефонного номера**
    * Создать исключение InvalidPhoneNumberException, если номер содержит недопустимые символы или его длина не равна 10 цифрам.
    * Класс PhoneNumberValidator должен содержать метод ValidatePhoneNumber(string phone), выполняющий проверку формата.
14. **Ограничение по весу багажа**
    * Создать исключение OverweightLuggageException, если вес багажа превышает 23 кг.
    * Класс Luggage должен иметь метод CheckWeight(int weight), проверяющий допустимый вес.
15. **Проверка срока годности продукта**
    * Создать исключение ExpiredProductException, если продукт просрочен.
    * Класс Product должен содержать метод CheckExpiration(DateTime expirationDate), сравнивающий дату истечения срока годности с текущей датой.

**Задание 2: Работа с внутренними исключениями и логированием**

**Общие требования:**

В каждой задаче необходимо:

1. Создать **класс, содержащий метод**, в котором вызывается сторонний метод (из другого класса).
2. Вызов метода должен приводить к возникновению исключения, которое перехватывается и **оборачивается во внутреннее исключение (InnerException)**.
3. **Логировать** всю информацию о возникшем исключении, включая стек вызовов и данные о внутреннем исключении.
4. Продемонстрировать работу с исключениями в Main через try-catch.

**Задачи:**

1. **Ошибка при чтении файла**
   * Класс FileReader содержит метод ReadFile(string path), который пытается открыть несуществующий файл и вызывает FileNotFoundException.
   * Класс FileProcessor вызывает этот метод и перехватывает исключение, оборачивая его в CustomFileException с InnerException.
2. **Ошибка при парсинге JSON**
   * Класс JsonParser содержит метод Parse(string json), который вызывает JsonException, если передана некорректная строка.
   * Класс DataProcessor вызывает этот метод, перехватывает исключение и логирует его в ParsingException.
3. **Ошибка при подключении к базе данных**
   * Класс DatabaseConnector содержит метод Connect(string connectionString), который выбрасывает SqlException при ошибке подключения.
   * Класс DatabaseManager перехватывает исключение и оборачивает его в DatabaseConnectionException.
4. **Ошибка при разборе строки в число**
   * Класс StringParser содержит метод ParseToInt(string input), который вызывает FormatException, если строка не может быть преобразована в число.
   * Класс NumberProcessor перехватывает исключение и логирует его в InvalidNumberException.
5. **Ошибка при отправке запроса в API**
   * Класс ApiClient содержит метод SendRequest(string url), который выбрасывает HttpRequestException, если сервер недоступен.
   * Класс RequestHandler перехватывает исключение и оборачивает его в ApiException.
6. **Ошибка при вычислении математической операции**
   * Класс MathOperations содержит метод Divide(int a, int b), который выбрасывает DivideByZeroException, если b == 0.
   * Класс CalculationManager перехватывает исключение и логирует его в MathException.
7. **Ошибка при загрузке конфигурационного файла**
   * Класс ConfigLoader содержит метод LoadConfig(string path), который вызывает FileNotFoundException, если файл не найден.
   * Класс ConfigurationManager перехватывает исключение и оборачивает его в ConfigurationException.
8. **Ошибка при работе с коллекциями**
   * Класс ListProcessor содержит метод GetElementAt(List<int> list, int index), который выбрасывает IndexOutOfRangeException, если индекс выходит за пределы списка.
   * Класс CollectionHandler перехватывает исключение и логирует его в CollectionException.
9. **Ошибка при десериализации XML**
   * Класс XmlDeserializer содержит метод Deserialize(string xml), который выбрасывает InvalidOperationException, если XML имеет неверный формат.
   * Класс XmlProcessor перехватывает исключение и оборачивает его в DeserializationException.
10. **Ошибка при работе с многопоточностью**

* Класс ThreadWorker содержит метод StartThread(Action action), который выбрасывает ThreadAbortException, если поток принудительно завершается.
* Класс ThreadManager перехватывает исключение и логирует его в ThreadOperationException.

1. **Ошибка при доступе к недопустимому объекту**

* Класс ObjectManager содержит метод AccessObject(object obj), который выбрасывает NullReferenceException, если объект null.
* Класс ObjectHandler перехватывает исключение и оборачивает его в ObjectAccessException.

1. **Ошибка при сериализации объекта**

* Класс Serializer содержит метод Serialize(object obj), который выбрасывает SerializationException, если объект не поддерживает сериализацию.
* Класс DataSerializer перехватывает исключение и логирует его в SerializationOperationException.

1. **Ошибка при отправке email-сообщения**

* Класс EmailSender содержит метод SendEmail(string email), который выбрасывает SmtpException, если сервер недоступен.
* Класс EmailManager перехватывает исключение и оборачивает его в EmailSendingException.

1. **Ошибка при доступе к защищённому файлу**

* Класс FileSecurity содержит метод OpenSecureFile(string path), который выбрасывает UnauthorizedAccessException, если доступ запрещён.
* Класс FileAccessManager перехватывает исключение и логирует его в SecurityException.

1. **Ошибка при генерации отчёта**

* Класс ReportGenerator содержит метод GenerateReport(), который выбрасывает IOException, если диск заполнен.
* Класс ReportManager перехватывает исключение и оборачивает его в ReportGenerationException.

**Задание 3: Генерация пользовательского исключения с помощью throw при определенных условиях**

**Общие требования:**

В каждой задаче необходимо:

1. **Создать пользовательское исключение** (наследник Exception), описывающее конкретную ошибку.
2. **Реализовать класс**, содержащий метод с проверкой входных данных или условий выполнения.
3. **При несоответствии условиям – выбрасывать пользовательское исключение** с помощью throw.
4. **Обработать исключение в Main** через try-catch.

**Задачи:**

1. **Проверка возраста для регистрации**
   * Класс UserRegistration содержит метод RegisterUser(int age).
   * Если возраст меньше 18 лет, выбрасывается AgeRestrictionException.
2. **Проверка баланса перед снятием средств**
   * Класс BankAccount содержит метод Withdraw(decimal amount).
   * Если сумма превышает баланс, выбрасывается InsufficientFundsException.
3. **Проверка строки на пустое значение**
   * Класс StringProcessor содержит метод ValidateInput(string input).
   * Если строка пустая или null, выбрасывается EmptyStringException.
4. **Проверка деления на ноль**
   * Класс Calculator содержит метод Divide(int a, int b).
   * Если b == 0, выбрасывается DivisionByZeroException.
5. **Проверка корректности email-адреса**
   * Класс EmailValidator содержит метод ValidateEmail(string email).
   * Если email не соответствует формату, выбрасывается InvalidEmailException.
6. **Проверка максимального лимита заказов**
   * Класс OrderManager содержит метод PlaceOrder(int itemCount).
   * Если itemCount > 100, выбрасывается OrderLimitExceededException.
7. **Проверка допустимого диапазона температур**
   * Класс TemperatureSensor содержит метод SetTemperature(int temp).
   * Если температура выходит за пределы [-50, 50], выбрасывается TemperatureOutOfRangeException.
8. **Проверка корректности пароля**
   * Класс PasswordManager содержит метод ValidatePassword(string password).
   * Если пароль короче 8 символов или не содержит цифр, выбрасывается WeakPasswordException.
9. **Проверка наличия товара на складе**
   * Класс Inventory содержит метод CheckStock(string item).
   * Если товар отсутствует, выбрасывается OutOfStockException.
10. **Проверка корректности формата даты**

* Класс DateValidator содержит метод ValidateDate(string date).
* Если дата не распознаётся, выбрасывается InvalidDateFormatException.

1. **Проверка уровня заряда аккумулятора**

* Класс BatteryManager содержит метод CheckBatteryLevel(int level).
* Если уровень заряда < 5%, выбрасывается LowBatteryException.

1. **Проверка лимита подключений к серверу**

* Класс ServerConnection содержит метод Connect(int activeConnections).
* Если подключений больше 10, выбрасывается TooManyConnectionsException.

1. **Проверка корректности номера телефона**

* Класс PhoneNumberValidator содержит метод ValidatePhoneNumber(string phone).
* Если номер не соответствует формату, выбрасывается InvalidPhoneNumberException.

1. **Проверка разрешённого времени доступа**

* Класс AccessControl содержит метод CheckAccessTime(int hour).
* Если время выходит за границы 9:00 - 18:00, выбрасывается AccessDeniedException.

1. **Проверка допустимого веса багажа в аэропорту**

* Класс LuggageChecker содержит метод CheckWeight(int weight).
* Если вес превышает 23 кг, выбрасывается ExcessLuggageException.