1. Назвіть та опишіть основні парадигми програмування
імперативне програмування

Це одна з найстаріших парадигм програмування. Вона має тісний зв'язок з архітектурою обчислювальної машини. Заснована на архітектурі фон Неймана. Працює, змінюючи стан програми за допомогою операторів присвоювання. Виконуються покрокові завдання, змінюючи стан. Основна увага приділяється тому, як досягти мети. Парадигма складається з декількох операторів і після виконання весь результат зберігається.

декларативне програмування

Парадигма ділиться на логіку, функціональність і базу даних. В інформатиці декларативне програмування - це стиль побудови програм, яка виражає логіку обчислень, не кажучи про потік управління. Вона часто розглядає програми як теорії деякої логіки. Це може спростити написання паралельних програм. Акцент робиться на тому, що потрібно зробити, а не на тому, як це повинно бути зроблено, в основному підкреслюючи, що написаний код дійсно робить. Вона просто оголошує бажаний результат, а не те, як він був проведений.

структурне програмування

Структурне програмування (так само модульне програмування) - це парадигма програмування, яка полегшує створення програм з читаним кодом і повторно використовуваними компонентами. Всі сучасні мови програмування підтримують структуроване програмування, але механізми підтримки, як і синтаксис мов програмування, розрізняються.

функціональне програмування

Функціональне програмування - це парадигма програмування, в якій ми намагаємося зв'язати все в стилі математичних функцій. Це декларативний тип стилю програмування. Його основний упор робиться на «що вирішувати» на відміну від імперативного стилю, де основний упор робиться на «як вирішувати». Він використовує вирази замість операторів. Вираз обчислюється для отримання значення, тоді як оператор виконується для присвоєння змінних.

логічне програмування

Цю парадигму можна назвати абстрактною моделлю обчислень. Вона вирішувала б логічні завдання, такі як головоломки, цикли і т. Д. У логічному програмуванні у нас є база знань, яку ми знаємо заздалегідь, і разом з питаннями, які передаються машині, вона дає результат. У нормальних мовах програмування така концепція бази знань недоступна, але при використанні концепції штучного інтелекту у нас є моделі, такі як модель сприйняття, яка використовує той же механізм.

Об'єктно-орієнтоване програмування

Програма написана як набір класів і об'єктів, призначених для взаємодії. Найменша і базова сутність - це об'єкт, і всі види обчислень виконуються тільки над об'єктами. Більше уваги приділяється даними, а не процедурою. ООП може впоратися практично з усіма видами реальних життєвих проблем, які існують сьогодні в скриптах.

• Компонентно-орієнтоване програмування

Стиль об'єктно-орієнтованого програмування, в основі якого лежить правило DRY - Do not Repeat Yourself (з англ. - не повторювати). Повторення в коді малоефективні і затратні за часом. Чим менше повторень, тим швидше йде розробка програми. З огляду на дедлайни, які найчастіше ставлять замовники, швидкість розробники є мало не однією з головних метрик. Якщо ви знаєте будь-якої front-end фреймворк, наприклад React, Angular або Vue, то вам повинно бути знайоме, як виглядає компонентно-орієнтована архітектура.

• прототипна-орієнтоване програмування

Це стиль об'єктно-орієнтованого програмування, в якому класи не визначаються явно, а є похідними шляхом додавання властивостей і методів до примірника іншого класу або, що рідше, додавання їх до порожнього об'єкту.
Простіше кажучи: цей тип стилю дозволяє створювати об'єкт без попереднього визначення його класу.

• Агентно-орієнтоване програмування

Першою реалізацією формалізму агентів була мова агентно-орієнтованого програмування AgentO. У цій мові агент описується в термінах набору здібностей (то, що може зробити агент), набору початкових зобов'язань і набору правил зобов'язань. Ключовим компонентом, який визначає, як агент діє, є набір правил, відповідних зобов'язань.

2. Дайте означення типу даних. Укажіть на які різновиди поділяють усі типи у мові C#

Однією з основних особливостей C# є те, що дана мова є строго типізований. А це означає, що кожна змінна і константа становить певний тип і даний тип строго визначений. Тип даних визначає діапазон значень, які може зберігати змінна або константа.

Отже, розглянемо систему вбудованих базових типів даних, яка використовується для створення змінних в C#. А вона представлена наступними типами.

boolean: зберігає значення true або false
byte: зберігає ціле число від -128 до 127 і займає 1 байт

short: зберігає ціле число від -32768 до 32767 і займає 2 байта

int: зберігає ціле число від -2147483648 до 2147483647 і займає 4 байта

long: зберігає ціле число від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 і займає 8 байт

double: зберігає число з плаваючою крапкою від ± 4.9 * 10-324 до ± 1.8 * 10308 і займає 8 байт

Як роздільник цілої та дробової частини в дрібних літералах використовується точка.

float: зберігає число з плаваючою крапкою від -3.4 * 1038 до 3.4 * 1038 і займає 4 байта

char: зберігає одиночний символ в кодуванні UTF-16 і займає 2 байта, тому діапазон збережених значень від 0 до 65535

При цьому змінна може приймати тільки ті значення, які відповідають її типу. Якщо змінна представляє цілочисельний тип, то вона не може зберігати дробові числа.

Цілі числа
Всі цілочисельні літерали, наприклад, числа 10, 4, -5, сприймаються як значення типу int, проте ми можемо привласнювати цілочисельні літерали іншим цілочисельним типам: byte, long, short. В цьому випадку C# автоматично здійснює відповідні перетворення:

Однак якщо ми захочемо привласнити змінної типу long дуже велике число, яке виходить за межі допустимих значень для типу int, то ми зіткнемося з помилкою під час компіляції:

Тут число 2147483649 є допустимим для типу long, але виходить за граничні значення для типу int. І так як все цілочисельні значення за замовчуванням розцінюються як значення типу int, то компілятор вкаже нам на помилку. Щоб вирішити проблему, треба додати до числа суфікс l або L, який вказує, що число представляє тип long:

Як правило, значення для цілочисельних змінних задаються в десятковій системі числення, однак ми можемо застосовувати і інші системи числення.
Для завдання шістнадцятирічного значення після символів 0x вказується число в шістнадцятковому форматі. Таким же чином вісімкове значення вказується після символу 0, а двоичное значення - після символів 0b.

Також цілі числа підтримують поділ розрядів числа за допомогою знака підкреслення:

Числа з плаваючою точкою

При присвоєнні змінної типу float дрібного литерала з плаваючою точкою, наприклад, 3.1, 4.5 і т.д., C# автоматично розглядає цей буквальний як значення типу double. І щоб вказати, що дане значення має розглядатися як float, нам треба використовувати суфікс f
І хоча в даному випадку обидві змінних мають практично одне значення, але ці значення будуть по-різному розглядатися і будуть займати різне місце в пам'яті.

Символи і рядки
Як значення змінна символьного типу отримує одиночний символ, укладений в одинарні лапки: char ch = 'e' ;. Крім того, змінної символьного типу також можна привласнити цілочисельне значення від 0 до 65535. У цьому випадку змінна знову ж буде зберігати символ, а цілочисельне значення буде вказувати на номер символу в таблиці символів Unicode (UTF-16).

Ще однією формою завдання символьних змінних є шестнадцатеричная форма: змінна отримує значення в шістнадцятковій формі, яке слід після символів "\ u". Наприклад, char ch = '\ u0066'; знову ж буде зберігати символ 'f'.

Символьні змінні не варто плутати із строковими, 'a' не ідентичне "a". Строкові змінні представляють об'єкт String, який на відміну від char або int не є примітивним типом в C#

Крім власне символів, які представляють букви, цифри, розділові знаки, інші символи, є спеціальні набори символів, які називають керуючими послідовностями. Наприклад, найпопулярніша послідовність - "\ n". Вона виконує перенесення на наступний рядок.

Починаючи з версії 15 C# підтримує тестові блоки (text blocks) - багаторядковий текст, одягнений у потрійні лапки. Розглянемо, у чому їх практична користь. За допомогою операції + ми можемо приєднати до одного тексту іншого, причому продовження тексту може розташовуватися на наступному рядку. Щоб при виведенні тексту відбувався перенос на наступний рядок, застосовується послідовність \ n. Весь текстовий блок обертається в потрійні лапки, при цьому не треба використовувати з'єднання рядків або послідовність \ n для їх перенесення.

3. Опишіть стилі об’єктно-орієнтованого програмування
Об'єктно-орієнтоване програмування

Програма написана як набір класів і об'єктів, призначених для взаємодії. Найменша і базова сутність - це об'єкт, і всі види обчислень виконуються тільки над об'єктами. Більше уваги приділяється даними, а не процедурою. ООП може впоратися практично з усіма видами реальних життєвих проблем, які існують сьогодні в скриптах.

• Компонентно-орієнтоване програмування

Стиль об'єктно-орієнтованого програмування, в основі якого лежить правило DRY - Do not Repeat Yourself (з англ. - не повторювати). Повторення в коді малоефективні і

затратні за часом. Чим менше повторень, тим швидше йде розробка програми. З огляду на дедлайни, які найчастіше ставлять замовники, швидкість розробники є мало не однією з головних метрик. Якщо ви знаєте будь-якої front-end фреймворк, наприклад React, Angular або Vue, то вам повинно бути знайоме, як виглядає компонентно-орієнтована архітектура.

• прототипна-орієнтоване програмування

Це стиль об'єктно-орієнтованого програмування, в якому класи не визначаються явно, а є похідними шляхом додавання властивостей і методів до примірника іншого класу або, що рідше, додавання їх до порожнього об'єкту.
Простіше кажучи: цей тип стилю дозволяє створювати об'єкт без попереднього визначення його класу.

• Агентно-орієнтоване програмування

Першою реалізацією формалізму агентів була мова агентно-орієнтованого програмування AgentO. У цій мові агент описується в термінах набору здібностей (то, що може зробити агент), набору початкових зобов'язань і набору правил зобов'язань. Ключовим компонентом, який визначає, як агент діє, є набір правил, відповідних зобов'язань.

4. Назвіть та опишіть принципи об’єктно-орієнтованого програмування (мінімум 3).
C# є об'єктно-орієнтованою мовою. Це означає, що писати програми на C# потрібно із застосуванням об'єктно-орієнтованого стилю. І стиль цей заснований на використанні в програмі об'єктів і класів.

Об'єктно-орієнтоване програмування (ООП) - це методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкування.

Основні принципи ООП:
абстракція
інкапсуляція
спадкування
поліморфізм

Абстракція - означає виділення значущої інформації і виключення з розгляду незначною. З точки зору програмування це правильний розподіл програми на об'єкти. Абстракція дозволяє відібрати головні характеристики і опустити другорядні.

Приклад: опис посад в компанії. Тут назва посади значима інформація, а опис обов'язків у кожної посади це другорядна інформація. Наприклад головною характеристикою для «директор» буде те, що це посада чимось керує, а чому саме

(директор по персоналу, фінансовий директор, виконавчий директор) це вже другорядна інформація.

Інкапсуляція - властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі. Для C# коректно буде говорити, що інкапсуляція це «приховування реалізації». Приклад з життя - пульт від телевізора. Ми натискаємо кнопочку «збільшити гучність» і вона збільшується, але в цей момент відбуваються десятки процесів, які приховані від нас. Для C#: можна створити клас з 10 методами, наприклад обчислюють площу складної фігури, але зробити з них 9 private. 10й метод буде називатися «вичіслітьПлощадь ()» і оголошений public, а в ньому вже будуть викликатися необхідні приховані від користувача методи. Саме його і буде викликати користувач.

Спадкування - властивість системи, що дозволяє описати новий клас на основі вже існуючого з частково або повністю запозичує функціональністю. Клас, від якого виробляється спадкування, називається базовим, батьківським або суперкласом. Новий клас - нащадком, спадкоємцем, дочірнім або похідним класом.

Поліморфізм - властивість системи використовувати об'єкти з однаковим інтерфейсом без інформації про тип і внутрішню структуру об'єкта.

5. Дайте означення масиву, що таке ранг масиву та які ранги масивів існують. Що таке ступінчасті масиви та коли їх доцільно використовувати.

Масив - це структура даних, в якій зберігаються елементи одного типу. Його можна уявити, як набір пронумерованих осередків, в кожну з яких можна помістити якісь дані (один елемент даних в одну клітинку). Доступ до конкретної осередку здійснюється через її номер. Номер елемента в масиві також називають індексом. У випадку з C# масив однорідний, тобто у всіх його елементах будуть зберігатися елементи одного типу. Так, масив цілих чисел містить тільки цілі числа (наприклад, типу int), масив рядків - тільки рядки, масив з елементів створеного нами класу Dog буде містити тільки об'єкти Dog. Тобто в C# ми не можемо помістити в перший осередок масиву ціле число, в другу String, а в третю - "собаку".

Створення масиву проводиться за допомогою наступної конструкції: new тип_даних [колічество_елементов], де new - ключове слово, що виділяє пам'ять для зазначеного в дужках кількості елементів. Наприклад, nums = new int [4]; - в цьому виразі створюється масив з чотирьох елементів int, і кожен елемент буде мати значення за замовчуванням - число 0.

Також можна відразу при оголошенні масиву форматувати його
Крім одновимірних масивів також бувають і багатовимірними. Найбільш відомий багатовимірний масив - таблиця, що представляє двомірний масив .

Багатовимірні масиви можуть бути також представлені як "зубчасті масиви". Ми можемо кожному елементу в двомірному масиві присвоїти окремий масив з різною кількістю елементів і отримати зубчатий масив

6. Назвіть та опишіть види поліморфізму

поліморфізм:

Поліморфізм - це здатність (в програмуванні) уявити той же інтерфейс для різних форм (типів даних), що лежать в основі. Це означає, що класи мають різну функціональність, незважаючи на те, що використовують загальний інтерфейс і можуть бути викликані динамічно через спеціальну посилання класу.

Класичним прикладом є клас Shape (фігура), і всі класи, які можуть успадковуватися від нього (квадрат, коло, додекаедр, неправильний багатокутник, знак і т.д.).

У цьому прикладі, кожен клас буде мати свою власну функцію Draw () і клієнтський код може виконувати наступні дії:

Shape shape = new Square ();
Shape.area () щоб отримати коректну поведінку для будь-якої форми.

Принадність поліморфізму в тому що код працює з різними класами і немає необхідності знати який клас його використовує, тому що всі вони працюють за одним принципом.

Процес, який використовується об'єктно-орієнтована мова програмування, який реалізує динамічний поліморфізм, називається динамічним зв'язуванням.

Примітка: Поліморфізм - це можливість вибору більш спеціалізованих методів в залежності від виконання викликаного об'єкта. Поліморфізм так само може використовуватися без участі будь-яких абстрактних класів.

переваги:

Створюється код, який використовується повторно: це означає, що якщо одного разу створені класи, реалізовані і протестовані, то вони можуть бути легко використані, не піклуючись про те, що написано в класі.
Це забезпечує більш загальний і слабо зв'язаної код.
Час компіляції значно зменшується, а розробка стає швидше.
Динамічне зв'язування: Один і той же інтерфейс може бути використаний для створення методів з різними реалізаціями.
Повна реалізація може бути замінена за допомогою сигнатури методу.

Перевизначення методу для досягнення поліморфізм: перевизначення інтерфейсів з двома методами: Один в батьківському класі, а інший в дочірньому класі з такими ж іменами і сигнатурами.

Перевизначення дозволяє визначити ту ж саму операцію по-різному для різних типів даних

наприклад:

```
while (it.hasNext ())
{
   Shape s = (Shape) it.next ();
   totalArea + = s.area (dim); // поліморфічне виклик методу. Буде викликаний правильний об'єкт.
}
```

Перевантаження методу або Спеціальний поліморфізм або статичний поліморфізм:

Перевантаження інтерфейсів з декількома методами в тому ж класі з тим же ім'ям, але з іншим тілом методу. Перевантаження методу дозволяє визначити ту ж операцію по-різному для різних даних. Якийсь час вона називалася статичним поліморфізмом, але насправді це не поліморфізм.

Перевантаження методів є не більше ніж двома методами, з однаковими іменами, але різними списками аргументів. Вона не має нічого спільного з наслідуванням і поліморфізмом. Перевантажений метод, це не те ж саме, що і перевизначення метод. Параметричний поліморфізм з використанням дженериків в C#:

При оголошенні класу, ім'я поля можна пов'язати з різними типами і ім'я методу так само можна асоціювати з різними параметрами і повертаються типами. C# підтримує параметричний поліморфізм з використанням дженериків.

Прикладом є список, який може приймати тип даних, що містять дженерекі.

```
List list = new ArrayList ();
```

7. Назвіть модифікатори доступу та опишіть їх

private (закритий) - доступ до члена класу не надається нікому, крім методів цього класу. Інші класи того ж пакета також не можуть звертатися до private-членам.

protected (захищений) - доступ в межах пакету і класів спадкоємців. Доступ в класі з іншого пакета буде до методів public і protected головного класу. Тобто якщо

package2.Class2 extends package1.MainClass, то всередині package2.Class2 методи з ідентифікатором protected з MainClass буде видно.

public (відкритий) - доступ для всіх з будь-якого іншого коду проекту

Модифікатори в списку розташовані по зростаючій видимості в програмі.

8. Що таке клас та об’єкт?

C# є об'єктно-орієнтованою мовою, тому такі поняття як "клас" і "об'єкт" грають в ньому ключову роль. Будь-яку програму на C# можна уявити як набір взаємодіючих між собою об'єктів.

Шаблоном або описом об'єкта є клас, а об'єкт являє екземпляр цього класу. Можна ще провести наступну аналогію. У нас у всіх є деяке уявлення про людину - наявність двох рук, двох ніг, голови, тулуба і т.д. Є деякий шаблон - цей шаблон можна назвати класом. Реально ж існуюча людина (фактично екземпляр даного класу) є об'єктом цього класу.

Клас визначається за допомогою ключового слова class Після назви класу йдуть фігурні дужки, між якими міститься тіло класу - тобто його поля і методи. Будь-який об'єкт може володіти двома основними характеристиками: стан - деякі дані, які зберігає об'єкт, і поведінка - дії, які може здійснювати об'єкт. Д
ля зберігання стану об'єкта в класі застосовуються поля або змінні класу. Для визначення поведінки об'єкта в класі застосовуються методи. Як правило, класи визначаються в різних файлах. В даному випадку для простоти ми визначаємо два класи в одному файлі. Варто відзначити, що в цьому випадку тільки один клас може мати модифікатор public (в даному випадку це клас Program), а сам файл коду повинен називатися по імені цього класу, тобто в даному випадку файл повинен називатися Program.C#.

Клас представляє новий тип, тому ми можемо визначати змінні, які представляють даний тип. Так, тут в методі main визначена змінна tom, яка представляє клас Person. Але поки ця змінна не вказує ні на який об'єкт і за замовчуванням вона має значення null. За великим рахунком ми її поки не можемо використовувати, тому спочатку необхідно створити об'єкт класу Person.

Конструктори Крім звичайних методів класи можуть визначати спеціальні методи, які називаються конструкторами. Конструктори викликаються при створенні нового об'єкта даного класу. Конструктори виконують ініціалізацію об'єкта. Якщо в класі не визначено жодного конструктора, то для цього класу автоматично створюється конструктор без параметрів. Якщо конструктор НЕ ініціалізує значення змінних об'єкта, то вони отримують значення за замовчуванням. Для змінних числових типів це число 0, а для типу string і класів - це значення null (тобто фактично відсутність значення). Якщо необхідно, щоб при створенні об'єкта проводилася якась логіка, наприклад, щоб поля класу отримували якісь певні значення, то можна визначити в класі свої конструктори.

Ключове слово this

Ключове слово this являє посилання на поточний екземпляр класу. Через це ключове слово ми можемо звертатися до змінних, методів об'єкта, а також викликати його конструктори.

Ініціалізатор

Крім конструктора початкову ініціалізацію об'єкта цілком можна було проводити за допомогою ініціалізатор об'єкта. Ініціалізатор виконується до будь-якого конструктора. Тобто в ініціалізатор ми можемо помістити код, загальний для всіх конструкторів

9. Опишіть поняття інтерфейсу та його властивостей у мові C#

Інтерфейси визначають деякий функціонал, який не має конкретної реалізації, який потім реалізують класи, які застосовують ці інтерфейси. І один клас може застосувати безліч інтерфейсів.

Щоб визначити інтерфейс, використовується ключове слово interface.

Інтерфейс може визначати константи і методи, які можуть мати, а можуть і не мати реалізації. Методи без реалізації схожі на абстрактні методи абстрактних класів. Так, в даному випадку оголошений один метод, який не має реалізації.

Всі методи інтерфейсу не мають модифікаторів доступу, але фактично за замовчуванням доступ public, так як мета інтерфейсу - визначення функціоналу для реалізації його класом. Тому весь функціонал повинен бути відкритий для реалізації.

Щоб клас застосував інтерфейс, треба використовувати ключове слово implements

При цьому треба враховувати, що якщо клас застосовує інтерфейс, то він повинен реалізувати всі методи інтерфейсу, як у випадку вище реалізований метод print. Потім в методі main ми можемо створити об'єкт класу Book і викликати його метод print. Якщо клас не реалізує якісь методи інтерфейсу, то такий клас повинен бути визначений як абстрактний, а його неабстрактне класи-спадкоємці потім повинні будуть реалізувати ці методи.

У той же час ми не можемо безпосередньо створювати об'єкти інтерфейсів

Однією з переваг використання інтерфейсів є те, що вони дозволяють додати в додаток гнучкості.

10. Назвіть різницю між інтерфейсом та абстрактним класом та випадки у яких доцільно використовувати кожен з них

Інтерфейси визначають деякий функціонал, який не має конкретної реалізації, який потім реалізують класи, які застосовують ці інтерфейси. І один клас може застосувати безліч інтерфейсів.

Щоб визначити інтерфейс, використовується ключове слово interface.

Інтерфейс може визначати константи і методи, які можуть мати, а можуть і не мати реалізації. Методи без реалізації схожі на абстрактні методи абстрактних класів. Так, в даному випадку оголошений один метод, який не має реалізації.

Всі методи інтерфейсу не мають модифікаторів доступу, але фактично за замовчуванням доступ public, так як мета інтерфейсу - визначення функціоналу для реалізації його класом. Тому весь функціонал повинен бути відкритий для реалізації.

Щоб клас застосував інтерфейс, треба використовувати ключове слово implements

При цьому треба враховувати, що якщо клас застосовує інтерфейс, то він повинен реалізувати всі методи інтерфейсу, як у випадку вище реалізований метод print. Потім в методі main ми можемо створити об'єкт класу Book і викликати його метод print. Якщо клас не реалізує якісь методи інтерфейсу, то такий клас повинен бути визначений як абстрактний, а його неабстрактне класи-спадкоємці потім повинні будуть реалізувати ці методи.

У той же час ми не можемо безпосередньо створювати об'єкти інтерфейсів
Однією з переваг використання інтерфейсів є те, що вони дозволяють додати в додаток гнучкості.

Крім звичайних класів в C# є абстрактні класи. Абстрактний клас схожий на звичайний клас. В абстрактному класі також можна визначити поля і методи, але в той же час не можна створити об'єкт або екземпляр абстрактного класу. Абстрактні класи покликані надавати базовий функціонал для класів-спадкоємців. А похідні класи вже реалізують цей функціонал.

При визначенні абстрактних класів використовується ключове слово abstract

Але головна відмінність полягає в тому, що ми не можемо використовувати конструктор абстрактного класу для створення його об'єкта.

Крім звичайних методів абстрактний клас може містити абстрактні методи. Такі методи визначаються за допомогою ключового слова abstract і не мають ніякої реалізації

Похідний клас зобов'язаний перевизначити і реалізувати всі абстрактні методи, які є в базовому абстрактному класі. Також слід враховувати, що якщо клас має хоча б один абстрактний метод, то даний клас повинен бути визначений як абстрактний.

Навіщо потрібні абстрактні класи? Припустимо, ми робимо програму для обслуговування банківських операцій і визначаємо в ній три класи: Person, який описує людини, Employee, який описує банківського службовця, і клас Client, який представляє клієнта банку. Очевидно, що класи Employee і Client будуть похідними від класу Person, так як обидва класи мають деякі загальні поля і методи. І так як всі об'єкти будуть представляти або співробітника, або клієнта банку, то безпосередньо ми від класу Person створювати об'єкти не будемо. Тому має сенс зробити його абстрактним.

Іншим хрестоматійним прикладом є система геометричних фігур. В реальності не існує геометричної фігури як такої. Є коло, прямокутник, квадрат, але просто фігури немає. Однак ж і коло, і прямокутник мають щось спільне і є фігурами

11. Охарактеризуйте поняття патернів програмування, опишіть 1 з них

Саме слово Патерн (pattern) з англійської перекладається як зразок, шаблон, приклад, екземпляр, заготовка. Дуже класне порівняння – це форма для випікання хліба. Раз зробили шаблон і використовуємо постійно для випікання усіх наступних хлібин.

Патерни Програмування – це напрацьовані ефективні підходи, техніки та правила вирішення задач при створенні програмного забезпечення. Вони не прив’язуються до певної мови програмування і можуть бути застосованими в основному незалежно від конкретної мови.

Також використовується цілий ряд інших назв як от: Шаблони Проектування Програмного Забезпечення, Дизайн Патерни (Design Patterns), Шаблони Програмування, і т.д.

Види Патернів Програмування
Є багато різноманітних категорій і видів патернів, від різноманітних програмістів. Але в даному курсі зупинимося лише на найбільш поширених патернах, які групуються у наступні три категорії:

Породжуючі – патерни, що надають рекомендації та техніки для створення нових об’єктів. Існує 5 породжуючих патернів.
Поведінкові – патерни, що надають рекомендації для реалізації тої чи іншої поведінки-функції існуючого об’єкта. Є 11 поведінкових патернів.
Структурні – даний тип патернів розглядає питання взаємодії між собою існуючих об’єктів. Всього є 7 структурних патернів.
Отже, всього маємо 23 патерни програмування

12. Дайте визначення виключенню, опишіть механізм обробки виключень
    Нерідко в процесі виконання програми можуть виникати помилки, при тому необов'язково з вини розробника. Деякі з них важко передбачити або передбачити, а іноді і зовсім неможливо. Так, наприклад, може несподівано обірватися з'єднання з мережею при передачі файлу. Подібні ситуації називаються винятками.

    У мові C# передбачені спеціальні засоби для обробки подібних ситуацій. Одним з таких засобів є конструкція try ... catch ... finally. При виникненні виключення в блоці try управління переходить в блок catch, який може обробити цей виняток. Якщо такого блоку не знайдено, то користувачеві відображається повідомлення про необробленому виключення, а подальше виконання програми зупиняється. І щоб такої зупинки не відбулося, і треба використовувати блок try..catch.
    При використанні блоку try ... catch спочатку виконуються всі інструкції між операторами try і catch. Якщо в блоці try раптом виникає виняток, то звичайний

порядок виконання зупиняється і переходить до інструкції catch. Тому коли виконання програми дійде до рядка numbers [4] = 45 ;, програма зупиниться і перейде до блоку catch

Вираз catch має наступний синтаксис: catch (тіп_ісключенія ім'я_змінної). В даному випадку оголошується змінна ex, яка має тип Exception. Але якщо виникло виключення не є винятком типу, зазначеного в інструкції catch, то воно не обробляється, а програма просто зависає або викидає повідомлення про помилку.

Але так як тип Exception є базовим класом для всіх винятків, то вираз catch (Exception ex) буде обробляти практично всі винятки. Обробка ж виключення в даному випадку зводиться до висновку на консоль стека трасування помилки за допомогою методу printStackTrace (), визначеного в класі Exception.

Після завершення виконання блоку catch програма продовжує свою роботу, виконуючи всі інші інструкції після блоку catch.

Конструкція try..catch також може мати блок finally. Однак цей блок необов'язковий, і його можна при обробці виключень опускати. Блок finally виконується в будь-якому випадку, чи виникло виключення в блоці try чи ні
Обробка декількох винятків
У C# є безліч різних типів виключень, і ми можемо розмежувати їх обробку, включивши додаткові блоки catch
Якщо у нас виникає виняток певного типу, то воно переходить до відповідного блоку catch.

оператор throw
Щоб повідомити про виконання виняткових ситуацій в програмі, можна використовувати оператор throw. Тобто за допомогою цього оператора ми самі можемо створити виняток і викликати його в процесі виконання.
За допомогою оператора throw за умовою викидається виключення. У той же час метод сам цей виняток не обробляє за допомогою try..catch, тому у визначенні методу використовується вираз throws Exception.
Базовим класом для всіх винятків є клас Throwable. Від нього вже успадковуються два класи: Error і Exception. Всі інші класи є похідними від цих двох класів.

Клас Error описує внутрішні помилки в виконуючою середовищі C#. Програміст має дуже обмежені можливості для обробки подібних помилок.

Власне виключення успадковуються від класу Exception. Серед цих винятків слід виділити клас RuntimeException. RuntimeException є базовим класом для так званої групи непроверяемих винятків (unchecked exceptions) - компілятор не перевіряє факт обробки таких винятків і їх можна не вказувати разом з оператором throws в оголошенні методу. Такі винятки є наслідком помилок розробника, наприклад, неправильне перетворення типів або вихід за межі масиву.

Деякі з класів непроверяемих винятків:

ArithmeticException: виключення, що виникає при розподілі на нуль

IndexOutOfBoundException: індекс поза межами масиву

IllegalArgumentException: використання невірного аргументу при виклику методу

NullPointerException: використання порожній посилання

NumberFormatException: помилка перетворення рядка в число

Всі інші класи, утворені від класу Exception, називаються перевіряються винятками (checked exceptions).

Деякі з класів перевіряються винятків:

CloneNotSupportedException: клас, для об'єкта якого викликається клонування, не реалізує інтерфейс Cloneable

InterruptedException: потік перерваний іншим потоком

ClassNotFoundException: неможливо знайти клас

Подібні винятки обробляються за допомогою конструкції try..catch. Або можна передати обробку методом, який буде викликати даний метод, вказавши виключення після оператора throws
Оскільки всі класи виключень успадковуються від класу Exception, то всі вони успадковують ряд його методів, які дозволяють отримати інформацію про характер винятку. Серед цих методів відзначимо найбільш важливі:

Метод getMessage () повертає повідомлення про виключення

Метод getStackTrace () повертає масив, що містить трасування стека винятку

Метод printStackTrace () відображає трасування стека

13. Охарактеризуйте поняття Upcasting та Downcasting
Оновлення та передача даних на C#
Процес перетворення одного типу даних на інший відомий як Typecasting та Upcasting, а Downcasting - це тип роздачі типів об’єктів. У C# об’єкт також можна вводити як тип даних. Батьківські та дочірні об’єкти - це два типи об’єктів. Отже, для об’єкта можливі два типи типової трансляції: батьківська дитина та дитина батьківська, або можна сказати Апскастинг та даундкастінг.

У C# об’єкт також можна вводити як тип даних. Батьківські та дочірні об’єкти - це два типи об’єктів. Отже, для об’єкта можливі два типи типової трансляції: батьківська дитина та дитина батьківська, або можна сказати Апскастинг та даундкастінг.

Приймання типів використовується для того, щоб переконатися, що змінні правильно обробляються функцією чи ні. У Upcasting і Downcasting ми вводимо одночасно дочірній об’єкт до батьківського об’єкта та батьківський об’єкт до дочірнього об’єкта одночасно. Ми можемо виконувати Upcasting неявно або явно, але downcast не може бути неявно можливим.
Давайте заглибимось у глибину обох цих типів об'єктів:

1) Модернізація
Оновлення даних - це тип розсилки типів об'єктів, при якому дочірній об'єкт переводиться на об'єкт батьківського класу. Використовуючи Upcasting, ми можемо легко отримати доступ до змінних та методів батьківського класу до дочірнього класу. Тут ми не маємо доступу до всіх змінних та методу. Ми отримуємо доступ лише до деяких зазначених змінних та методів дочірнього класу. Модернізація також відома як узагальнення та розширення.

2) Передача даних
Оновлення даних - це ще один тип розсилки об'єктів. У Upcasting ми призначаємо дочірньому класу довідковий об'єкт батьківського класу. У C# ми не можемо присвоїти дочірньому класу довідковий об'єкт батьківського класу, але якщо ми виконуємо передачу даних, ми не отримаємо жодної помилки часу компіляції. Однак, коли ми запускаємо його, він видає "ClassCastException". Тепер справа в тому, що якщо передавання даних неможливе на C#, то чому це дозволено компілятором? У C# деякі сценарії дозволяють нам виконувати передачу даних. Тут об’єкт підкласу посилається батьківським класом.
14. Охарактеризуйте тип даних String, та опишіть операції з рядками.
Рядок являє собою послідовність символів. Для роботи з рядками в C# визначено клас String, який надає ряд методів для маніпуляції рядками. Фізично об'єкт String є посилання на область в пам'яті, в якій розміщені символи.

Для створення нового рядка ми можемо використовувати один з конструкторів класу String, або безпосередньо привласнити рядок в подвійних лапки

При роботі з рядками важливо розуміти, що об'єкт String є незмінним (immutable). Тобто при будь-яких операціях над рядком, які змінюють цей рядок, фактично буде створюватися новий рядок.

Оскільки рядок розглядається як набір символів, то ми можемо застосувати метод length () для знаходження довжини рядка або довжини набору символів

А за допомогою методу toCharArray () можна назад перетворити рядок в масив символів

Рядок може бути порожньою. Для цього їй можна присвоїти порожні лапки або видалити з стоки все символи

У цьому випадку довжина рядка, яка повертається методом length (), дорівнює 0.

Клас String має спеціальний метод, який дозволяє перевірити рядок на порожнечу - isEmpty (). Якщо рядок порожня, він повертає true

Мінлива String може не вказувати на який-небудь об'єкт і мати значення null

Значення null не еквівалентне порожній рядку.
Так як змінна не вказує ні на який об'єкт String, то відповідно ми не можемо звертатися до методів об'єкта String. Щоб уникнути подібних помилок, можна попередньо перевіряти рядок на null

Основні методи класу String
Основні операції з рядками розкривається через методи класу String, серед яких можна виділити наступні:

concat (): об'єднує рядки

valueOf (): перетворює об'єкт в строковий вид

join (): з'єднує рядки з урахуванням роздільник

compareTo (): порівнює два рядки

charAt (): повертає символ рядка за індексом

getChars (): повертає групу символів

equals (): порівнює рядки з урахуванням регістру

equalsIgnoreCase (): порівнює рядки без урахування регістру

regionMatches (): порівнює підрядка в рядках

indexOf (): знаходить індекс першого входження підрядка в рядок

lastIndexOf (): знаходить індекс останнього входження підрядка в рядок

startsWith (): визначає, чи починається рядок з підрядка

endsWith (): визначає, чи закінчується рядок на певну підрядок

replace (): замінює в рядку одну подстроку на іншу

trim (): видаляє початкові і кінцеві пробіли

substring (): повертає підрядок, починаючи з певного індексу до кінця або до певного індексу

toLowerCase (): переводить всі символи рядка в нижній регістр

toUpperCase (): переводить всі символи рядка в верхній регістр

15. Охарактеризуйте поняття Generics(Узагальнення) та їх застосування у мові C#

Узагальнення - це параметризовані типи. З їх допомогою можна оголошувати класи, інтерфейси і методи, де тип даних вказано у вигляді параметра. Узагальнення додали в мову безпеку типів.

Узагальнення – це механізм побудови програмного коду для деякого типу з довільним іменем з метою його подальшого конвертування (перетворення) у будь-який конкретний посилальний тип. Реалізацію конвертування з узагальненого типу в деякий конкретний здійснює компілятор.

Узагальнення можуть бути застосовані до класів, інтерфейсів чи методів. Якщо клас, інтерфейс чи метод оперує деяким узагальненим типом T, то цей клас (інтерфейс, метод) називається узагальненим. Тип, що отримує узагальнений клас в якості параметру називається параметризованим типом. Ім’я параметризованого типу можна задавати будь-яким (T, Type, TT і т.д.).

Використання узагальнень в мові C# дає наступні переваги:

- забезпечується компактність програмного коду;
- завдяки узагальненням усі операції приведення типів виконуються автоматично і неявно. Повторно використовуваний код обробляється більш легше та безпечніше;
- узагальнення забезпечують типову безпеку типів на відміну від використання посилань на тип Object.

Як відомо, у мові C# усі класи чи інтерфейси є підкласами класу Object. Це означає, що з допомогою посилання на клас Object можна оперувати різнотипними об’єктами. Однак, такий спосіб не забезпечує типової безпеки типів.

16. Охарактеризуйте поняття документування коду та code convention C#

Документація програмного забезпечення(англ. *software documentation*) - супроводжуючі документи до програмного забезпечення, які містять в собі інформацію, що описує загальні положення необхідні для ознайомлення перед тим як використовувати його за призначенням. Така документація дуже важлива і описує не тільки яким чином правильно використовувати поставлене програмне забезпечення, а й пояснює основні використані алгоритми. В залежності від складності кожного окремого програмного забезпечення, його специфіки, а також ліцензії під якою воно створене - документація може варіюватися за обсягом і за змістом.

Основні компоненти

- специфікація - перелік і призначення всіх файлів програмного виробу, включаючи файли документації;

- відомість власників оригіналів - список підприємств, які зберігають оригінали програмних документів, складається тільки для складних програмних виробів;
- текст програми - запис кодів програми та коментарі до них;
- опис програми - інформація про логічну структуру та функціонування програми;
- програма і методика випробувань - перелік і опис вимог, які повинні бути перевірені в ході випробування програми, методи контролю;
- технічне завдання - документ, в якому викладаються призначення і область застосування програми, вимоги до програмного виробу, стадії і терміни розробки, види випробувань;
- пояснювальна записка - обґрунтування прийнятих і застосованих технічних і техніко-економічних рішень, схеми та опис алгоритмів, загальний опис роботи програмного виробу;

До програмних документів віднесено також документи, що забезпечують функціонування та експлуатацію програм - експлуатаційні документи:

- відомість експлуатаційних документів - містить список експлуатаційних документів на програмний виріб, до яких відносяться формуляр, опис застосування, керівництво системного програміста, керівництво програміста, керівництво оператора, опис мови, керівництво з технічного обслуговування;
- формуляр - містить основні характеристики програмного виробу, склад і відомості про експлуатацію програми;
- опис застосування - містить інформацію про призначення та галузі застосування програмного виробу, обмеження при застосуванні, клас і методи вирішуваних завдань, конфігурацію технічних засобів;
- керівництво системного програміста - містить відомості для перевірки, настроювання і функціонування програми при конкретному застосуванні;
- керівництво програміста - містить відомості для експлуатації програмного виробу;
- керівництво оператора - містить докладну інформацію для користувача, який забезпечує його спілкування з ЕОМ у процесі виконання програми;
- опис мови - містить синтаксис і семантику мови;
- керівництво з технічного обслуговування - містить відомості для застосування тестових і діагностичних програм при обслуговуванні технічних засобів.

Всього існує чотири рівня документації:

– Архітектурна (проектна) – наприклад, дизайн-специфікація. Це документи, що описують моделі, методології, інструменти та засоби розробки, обрані для даного проекту.

– Технічна – вся документація, що супроводжує розробку. Сюди входять різноманітні документи, що пояснюють розробку системи на рівні окремих модулів. Як правило, пишеться у вигляді коментарів до джерела коду, які потім структуруються у вигляді HTML-документів.

– Користувацька – включає довідникові та пояснюючі матеріали, необхідні кінцевому користувачу для роботи з системою. Це, наприклад, Readme та Userguide, розділ довідки по програмі.

– Маркетингова – включає рекламні матеріали, які супроводжують випуск продукту. Її мета – в яскравій формі презентувати функціональність та конкурентні переваги продукту.

Code Convention :
https://translatedby.com/you/C#-code-conventions/into-ru/
https://skillbox.ru/media/base/C#_code_style_kak_pravilno_oformlyat_kod_C#/
https://habr.com/ru/post/112042/

17. Опишіть що таке тестування коду та назвіть його основні види, опишіть один з них.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок [17].

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений професором Едсжером Дейкстри у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!".

Існує безліч підходів до вирішення завдання тестування та верифікації ПЗ, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких.

З точки зору ISO 9126 якість (програмних засобів) можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових: надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність.

Більш повний список атрибутів і критеріїв можна знайти в стандарті ISO 9126 Міжнародної організації зі стандартизації. Склад і зміст документації, супутньої процесу тестування, визначається стандартом IEEE 829-1998 Standard for Software Test Documentation.
Існує кілька ознак, за якими класифікують тестування програмного забезпечення на види тестування.

За об'єктом тестування виділяють наступні види тестування ПЗ:

- функціональне тестування **(functional testing);**
- дослідницьке тестування **(exploratory testing);**
- тестування продуктивності **(performance testing);**
- навантажувальне тестування **(load testing);**
- димне тестування **(smoke testing);**
- стрес-тестування **(stress testing);**
- тестування стабільності **(stability / endurance / soak testing);**
- тестування зручності використання **(usability testing);**
- тестування інтерфейсу користувача **(ui testing);**
- тестування безпеки **(security testing);**
- тестування локалізації **(localization testing);**
- тестування сумісності **(compatibility testing).**

У залежності від переслідуваних цілей, види тестування можна умовно розділити на наступні типи:

- функціональне тестування **(functional);**
- нефункціональне тестування **(non-functional);**
- тестування пов'язане зі змінами.

За знанням системи:

- тестування чорної скриньки **(black box);**
- тестування білої скриньки **(white box);**
- тестування сірої скриньки **(gray box).**

За ступенем автоматизації:

- ручне тестування **(manual testing);**
- автоматизоване тестування **(automated testing);**
- напівавтоматизоване тестування **(semiautomated testing).**

За ступенем ізольованості компонентів:

- компонентне (модульне) тестування **(component / unit testing);**
- інтеграційне тестування **(integration testing);**
- системне тестування **(system / end-to-end testing).**

За часом проведення тестування:

- Альфа-тестування **(alpha testing):**
- тестування нової функціональності **(new feature testing);**
- регресійне тестування **(regression testing);**
- тестування при здачі **(acceptance testing);**
- Бета-тестування **(beta testing).**

18. Опишіть організувати вивід символьних та числових значень на екран, та як організувати форматований вивід

Найбільш простий спосіб взаємодії з користувачем представляє консоль: ми можемо виводити на консоль деяку інформацію або, навпаки, зчитувати з консолі деякі дані. Для взаємодії з консоллю в C# застосовується клас System, а його функціональність власне забезпечує консольне введення і виведення.

Висновок на консоль

Для створення потоку виведення в клас System визначено об'єкт out. В цьому об'єкті визначено метод println, який дозволяє вивести на консоль деяке значення з подальшим переведенням курсору консолі на наступний рядок.

У метод println передається будь-яке значення, як правило, рядок, яке треба вивести на консоль.

При необхідності можна і не переводити курсор на наступний рядок. В цьому випадку можна використовувати метод System.out.print (), який аналогічний println за тим винятком, що не чинить перекладу на наступний рядок.

Консольний висновок даної програми:

Але за допомогою методу System.out.print також можна здійснити переклад каретки на наступний рядок. Для цього треба використовувати escape-послідовність \ n

Нерідко необхідно підставляти в рядок якісь дані. Наприклад, у нас є два числа, і ми хочемо вивести їх значення на екран.

Але в C# є також функція для форматованого виведення, успадкована від мови C: System.out.printf ().

В даному випадку символи% d позначають специфікатор, замість якого підставляє один з аргументів. Специфікаторів і відповідних їм аргументів може бути безліч. В даному випадку у нас тільки два аргументи, тому замість першого% d підставляє значення змінної x, а замість другого - значення змінної y. Сама буква d означає, що даний специфікатор буде використовуватися для виведення цілочисельних значень.

Крім специфікатор% d ми можемо використовувати ще ряд специфікаторів для інших типів даних:

% X: для виведення шістнадцяткових чисел

% F: для виведення чисел з плаваючою точкою

% E: для виведення чисел у експоненційної формі, наприклад, 1.3e + 01

% C: для виведення одиночного символу

% S: для виведення строкових значень

При виведенні чисел з плаваючою точкою ми можемо вказати кількість знаків після коми, для цього використовуємо специфікатор на% .2f, де .2 вказує, що після коми буде два знака. У підсумку ми отримаємо наступний висновок:

Name: Tom Age: 30 Height: 1,70
Введення з консолі
Для отримання введення з консолі в класі System визначено об'єкт in. Однак безпосередньо через об'єкт System.in не дуже зручно працювати, тому, як правило, використовують клас Scanner, який, в свою чергу використовує System.in.
Так як клас Scanner знаходиться в пакеті C#.util, то ми спочатку його імпортуємо з допомогою інструкції import C#.util.Scanner.

Для створення самого об'єкта Scanner в його конструктор передається об'єкт System.in. Після цього ми можемо отримувати вводяться значення. Наприклад, в даному випадку спочатку виводимо запрошення до вводу і потім отримуємо вводиться число в змінну num.

Щоб отримати введене число, використовується метод in.nextInt () ;, який повертає введене з клавіатури цілочисельне значення.

Клас Scanner має ще ряд методів, які дозволяють отримати введені користувачем значення:

next (): зчитує введений рядок до першого пробілу

nextLine (): зчитує всю введену рядок

nextInt (): зчитує введене число int

nextDouble (): зчитує введене число double

nextBoolean (): зчитує значення boolean

nextByte (): зчитує введене число byte

nextFloat (): зчитує введене число float

nextShort (): зчитує введене число short

Тобто для введення значень кожного примітивного типу в класі Scanner визначений свій метод.

Зверніть увагу, що для введення значення типу float (те ж саме відноситься до типу double) застосовується число "1,7", де роздільником є кома, а не "1.7", де роздільником є точка. В даному випадку все залежить від поточної мовної локалізації системи. У моєму випадку російськомовна локалізація, відповідно вводити необхідно числа, де роздільником є кома. Те ж саме стосується багатьох інших локалізацій, наприклад, німецької, французької і т.д., де застосовується кома.

19. Опишіть форми умовного оператора у мові C# та його загальний вигляд у повній та скороченій формах.

Одним з фундаментальних елементів багатьох мов програмування є умовні конструкції. Дані конструкції дозволяють спрямувати роботу програми по одній із колій в залежності від певних умов.

У мові C# використовуються наступні умовні конструкції: if..else і switch..case

Конструкція if / else
Вираз if / else перевіряє істинність деякого умови і залежно від результатів перевірки виконує певний код
Після ключового слова if ставиться умова. І якщо ця умова виконується, то спрацьовує код, який поміщений в далі в блоці if після фігурних дужок. В якості умов виступає операція порівняння двох чисел.

Так як, в даному випадку перше число більше другого, то вираз num1> num2 істинно і повертає значення true. Отже, управління переходить в блок коду після фігурних дужок і починає виконувати містяться там інструкції, а саме метод System.out.println ("Перше число більше другого") ;. Якби перше число виявилося б менше другого або дорівнює йому, то інструкції в блоці if не виконувалися б.

Але що, якщо ми захочемо, щоб при недотриманні умови також виконувалися будь-які дії? У цьому випадку ми можемо додати блок else
Але при порівнянні чисел ми можемо нарахувати три стану: перше число більше другого, перше число менше другого і числа рівні. За допомогою виразу else if, ми можемо обробляти додаткові умови
Також ми можемо поєднати відразу кілька умов, використовуючи логічні оператори

конструкція switch
Конструкція switch / case аналогічна конструкції if / else, так як дозволяє обробити відразу кілька умов
Після ключового слова switch в дужках йде сравниваемое вираз. Значення цього виразу послідовно порівнюється зі значеннями, поміщеними після операторів case. І якщо збіг знайдено, то буде виконує відповідний блок case.

В кінці блоку case ставиться оператор break, щоб уникнути виконання інших блоків.
тернарного операція

Тернарного операція має наступний синтаксис: [перший операнд - умова]? [Другий операнд]: [третій операнд]. Таким чином, в цій операції беруть участь відразу три операнда. Залежно від умови тернарного операція повертає другий або третій операнд: якщо умова одно true, то повертається другий операнд; якщо умова одно false, то третій.

20. Опишіть що таке оператор циклу, основні складові та види циклів.

Цикли являються одним з видів керуючих конструкцій. Цикли дозволяють в залежності від певних умов виконувати певну дію безліч разів. У мові C# є такі види циклів:

for

while

do ... while

цикл for

Цикл for має наступне формальне визначення:

```
for ([ініціалізація лічильника]; [умова]; [зміна лічильника])
{
   // дії
}
```

Розглянемо стандартний цикл for:

```
for (int i = 1; i <9; i ++) {
   System.out.printf ( "Квадрат числа% d дорівнює% d \ n", i, i * i);
}
```

Перша частина оголошення циклу - int i = 1 створює і ініціалізує лічильник i. Лічильник необов'язково повинен представляти тип int. Це може бути і будь-який інший числовий тип, наприклад, float. Перед виконанням циклу значення лічильника буде дорівнює 1. В даному випадку це те ж саме, що і оголошення змінної.

Друга частина - умова, при якому буде виконуватися цикл. В даному випадку цикл буде виконуватися, поки i не досягне 9.

І третя частина - приріст лічильника на одиницю. Знову ж нам необов'язково збільшувати на одиницю. Можна зменшувати: i--.

В результаті блок циклу спрацює 8 разів, поки значення і не стане рівним 9. І кожен раз це значення буде збільшуватися на 1.

Нам необов'язково вказувати всі умови при оголошенні циклу.
Визначення циклу залишилося тим же, тільки тепер блоки у визначенні у нас порожні: for (;;). Тепер немає ініціалізованої змінної-лічильника, немає умови, тому цикл буде працювати вічно - нескінченний цикл.

Або можна опустити ряд блоків:

Цей приклад еквівалентний першому прикладу: у нас також є лічильник, тільки створений він поза циклом. У нас є умова виконання циклу. І є приріст лічильника вже в самому блоці for.

Цикл for може визначати відразу кілька змінних і управляти ними
цикл do
Цикл do спочатку виконує код циклу, а потім перевіряє умова в інструкції while. І поки ця умова істинно, цикл повторюється. наприклад:

1
2
3
4
5
6
```
int j = 7;
do {
   System.out.println (j);
   j--;
}
while (j> 0);
```
В даному випадку код циклу спрацює 7 разів, поки j не опиниться рівним нулю.
Важливо відзначити, що цикл do гарантує хоча б одноразове виконання дій, навіть якщо умова в інструкції while НЕ було це слово. Т

цикл while
Цикл while відразу перевіряє істинність деякого умови, і якщо умова істинна, то код циклу виконується

Оператори continue і break
Оператор break дозволяє вийти з циклу в будь-який його момент, навіть якщо цикл не закінчив свою роботу:

наприклад:

1
2

```
3
4
5
for (int i = 0; i <10; i ++) {
   if (i == 5)
      break;
   System.out.println (i);
}
```

Коли лічильник стане рівним 5, спрацює оператор break, і цикл завершиться.

Тепер зробимо так, щоб якщо число дорівнює 5, цикл не завершувався, а просто переходив до наступної ітерації. Для цього використовуємо оператор continue

У цьому випадку, коли виконання циклу дійде до числа 5, програма просто пропустить це число і перейде до наступного.

21. Опишіть загальний вигляд оператора вибору switch та яке призначення операторів break і continue

конструкція switch

Конструкція switch / case аналогічна конструкції if / else, так як дозволяє обробити відразу кілька умов

Після ключового слова switch в дужках йде сравниваемое вираз. Значення цього виразу послідовно порівнюється зі значеннями, поміщеними після операторів case. І якщо збіг знайдено, то буде виконує відповідний блок case.

22. Охарактеризуйте поняття регулярних виразів та випадки їх застосування. Опишіть синтаксис регулярних виразів.

Регулярні вирази потужний інструмент для обробки рядків.

Регулярні вирази дозволяють задати шаблон, якому повинна відповідати рядок або підрядок. Деякі методи класу String приймають регулярні вирази і використовують їх для виконання операцій над рядками.

split Для поділу стрічки на підрядка застосовується метод split (). Як параметр він може приймати регулярний вираз, яке представляє критерій поділу стрічки. Для поділу застосовується регулярне вираз "\\ s * (\\ s |, |! | \\.) \\ s *". Подвиражніе "\\ s" по суті являє пробіл. Зірочка вказує, що символ може бути присутнім від 0 до нескінченної кількості разів. Тобто додаємо зірочку і ми отримуємо невизначену кількість йдуть підряд прогалин - "\\ s *" (тобто неважливо, скільки прогалин між словами). Причому прогалини може взагалі не бути. У дужках вказує група виразів, яка може йти після невизначеної кількості прогалин. Група дозволяє нам визначити званих вели значень через вертикальну риску, і подрядок повинна відповідати одному з цих значень. Тобто в групі "\\ s |, |! | \\." Подрядок може відповідати пробілу, коми, знаку оклику або точці. Причому оскільки точка являє спеціальну послідовність, то, щоб вказати, що ми маємо на увазі саме знак точки, а не спеціальну послідовність, перед точкою ставимо слеш.

Відповідність рядка. matches Ще один метод класу String - matches () приймає регулярний вираз і повертає true, якщо рядок відповідає цьому виразу. Інакше повертає false. Наприклад, перевіримо, чи відповідає рядок номеру телефону В даному випадку в регулярному вираз першу чергу визначається група "(\\ + *)". Тобто спочатку може йти

знак плюса, але також він може бути відсутнім. Далі дивимося, чи відповідають наступні 11 символів цифрам. Вираз "\\ d" представляє цифровий символ, а число в фігурних дужках - {11} - скільки разів даний тип символів повинен повторюватися. Тобто ми шукаємо рядок, де спочатку може йти знак плюс (або він може бути відсутнім), а потім йде 11 цифрових символів.

клас Pattern Велика частина функціональності по роботі з регулярними виразами в C# зосереджена в пакеті C#.util.regex. Саме регулярний вираз являє шаблон для пошуку збігів в рядку. Для завдання подібного шаблону і пошуку подстрок в рядку, які задовольняють даним шаблоном, в C# визначені класи Pattern і Matcher. Для простого пошуку відповідників в класі Pattern визначено статичний метод boolean matches (String pattern, CharSequence input). Даний метод повертає true, якщо послідовність символів input повністю відповідає шаблоном рядки pattern Але, як правило, для пошуку відповідників застосовується інший спосіб - використання класу Matcher.

клас Matcher Розглянемо основні методи класу Matcher:

boolean matches (): повертає true, якщо вся рядок збігається з шаблоном

boolean find (): повертає true, якщо в рядку є підрядок, яка збігається з шаблоном, і переходить до цієї групи

String group (): повертає підрядок, яка збіглася з шаблоном в результаті виклику методу find. Якщо збіг відсутня, то метод генерує виняток IllegalStateException.

int start (): повертає індекс поточного збіги

int end (): повертає індекс наступного збігу після поточного

String replaceAll (String str): замінює всі знайдені збіги підрядком str і повертає змінену рядок з урахуванням замін

Використовуємо клас Matcher. Для цього спочатку треба створити об'єкт Pattern за допомогою статичного методу compile (), який дозволяє встановити шаблон Як шаблон виступає рядок "Hello".

Метод compile () повертає об'єкт Pattern, який ми потім можемо використовувати в програмі. У класі Pattern також визначено метод matcher (String input), який в якості параметра приймає рядок, де треба проводити пошук, і повертає об'єкт Matcher Потім у об'єкта Matcher викликається метод matches () для поїї