Національний університет «Львівська Політехніка»

Інститут комп’ютерних технологій, автоматики та метрології

Кафедра електронних обчислювальних машин



Звіт

Про виконання лабораторної роботи №6

З дисципліни «Кросплатформлені засоби програмування»

**Виконав:**

студент групи КІ-305

Панасюк Д.С.

**Перевірив:**

Доцент кафедри ЕОМ

Іванов Ю. С.

Львів – 2023

**Тема:** Параметризоване програмування.

**Мета:** Оволодіти навиками параметризованого програмування мовою Java.

**Завдання:**

1. Створити параметризований клас, що реалізує предметну область задану варіантом.Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.  
2. Автоматично згенерувати документацію до розробленого пакету.  
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.  
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її  
виконання та фрагменту згенерованої документації та завантажити його у ВНС.  
5. Дати відповідь на контрольні запитання.

**Завдання згідно варіанту:**

**Варіант: 21**

**Виконання:**

**Код програми:**

**package** KI305.Panasiuk.Lab7;

/\*\*

\* A program driver to demonstrate the use of the TrashCan class and placing items in it.

\*

\* **@author** Taras Panasiuk

\* **@version** 1.0

\*/

**public** **class** TrashCanDriver {

/\*\*

\* The main method of the program.

\*

\* **@param** args Command-line arguments (not used in this application).

\*/

**public** **static** **void** main(String[] args) {

// Create a TrashCan container for various types of trash

TrashCan<Item> trashCan = **new** TrashCan<>();

// Create objects of the Paper and PlasticBottle classes and add them to the container

Paper paper1 = **new** Paper("Contents of the first paper");

Paper paper2 = **new** Paper("Contents of the second paper");

PlasticBottle bottle1 = **new** PlasticBottle(500);

PlasticBottle bottle2 = **new** PlasticBottle(750);

PlasticBottle bottle3 = **new** PlasticBottle(500);

PlasticBottle bottle4 = **new** PlasticBottle(750);

Paper paper = **new** Paper("Contents of the second paper");

trashCan.add(paper1);

trashCan.add(paper2);

trashCan.add(bottle1);

trashCan.add(bottle2);

trashCan.add(bottle3);

trashCan.add(bottle4);

trashCan.add(paper);

System.***out***.println(TrashCan.*count\_paper*);

System.***out***.println(TrashCan.*count\_plastic*);

}

}

**package** KI305.Panasiuk.Lab7;

/\*\*

\* An interface representing an item that can be placed in a trash can.

\* Implementing classes must provide methods to get the size of the item and to print its details.

\* This interface extends the Comparable interface to allow for comparison between items.

\* **@author** Taras Panasiuk

\* **@version** 1.0

\*/

**public** **interface** Item **extends** Comparable<Item> {

/\*\*

\* Gets the size of the item.

\*

\* **@return** The size of the item.

\*/

**public** **int** getSize();

/\*\*

\* Prints the details of the item.

\*/

**public** **void** print();

}

**package** KI305.Panasiuk.Lab7;

**import** java.util.Stack;

/\*\*

\* A generic class representing a trash can that can hold items of type T.

\* It provides methods for adding, removing, checking emptiness, getting size, showing contents, clearing, and peeking into the trash can.

\*

\* **@param** <T> The type of items that the trash can can hold, must implement the Item interface.

\* **@author** Taras Panasiuk

\* **@version** 1.0

\*/

**public** **class** TrashCan<T **extends** Item> {

**private** Stack<T> items;

**public** **static** **int** *count\_paper* = 0;

**public** **static** **int** *count\_plastic* = 0;

/\*\*

\* Constructs an empty trash can.

\*/

**public** TrashCan() {

items = **new** Stack<T>();

}

/\*\*

\* Adds an item to the trash can.

\*

\* **@param** item The item to be added.

\*/

**public** **void** add(T item) {

items.push(item);

**if** (item.getClass() == Paper.**class**) {

*count\_paper* += 1;

}

**if** (item.getClass() == PlasticBottle.**class**){

*count\_plastic* += 1;

}

}

/\*\*

\* Removes and returns the top item from the trash can.

\*

\* **@return** The top item removed from the trash can, or null if the trash can is empty.

\*/

**public** T remove() {

**if** (!items.isEmpty()) {

**return** items.pop();

}

**return** **null**;

}

/\*\*

\* Checks if the trash can is empty.

\*

\* **@return** true if the trash can is empty, false otherwise.

\*/

**public** **boolean** isEmpty() {

**return** items.isEmpty();

}

/\*\*

\* Gets the size of the trash can.

\*

\* **@return** The number of items in the trash can.

\*/

**public** **int** size() {

**return** items.size();

}

/\*\*

\* Displays the contents of the trash can by printing each item.

\*/

**public** **void** showContents() {

**for** (T item : items) {

item.print();

}

}

/\*\*

\* Clears all items from the trash can.

\*/

**public** **void** clear() {

items.clear();

}

/\*\*

\* Peeks at the top item in the trash can without removing it.

\*

\* **@return** The top item in the trash can, or null if the trash can is empty.

\*/

**public** T peek() {

**if** (!items.isEmpty()) {

**return** items.peek();

}

**return** **null**;

}

}

**package** KI305.Panasiuk.Lab7;

/\*\*

\* A class representing a piece of paper that implements the Item interface.

\* It provides methods to get the content of the paper and to print its details.

\* This class also implements the Comparable interface for comparison with other items.

\*

\* **@author** Taras Panasiuk

\* **@version** 1.0

\*/

**public** **class** Paper **implements** Item {

**private** String content;

/\*\*

\* Constructs a piece of paper with the specified content.

\*

\* **@param** content The content of the paper.

\*/

**public** Paper(String content) {

**this**.content = content;

}

/\*\*

\* Gets the content of the paper.

\*

\* **@return** The content of the paper.

\*/

**public** String getContent() {

**return** content;

}

@Override

**public** **int** getSize() {

**return** content.length();

}

@Override

**public** **void** print() {

System.***out***.println("Paper Content: " + content);

}

@Override

**public** **int** compareTo(Item other) {

**if** (other **instanceof** Paper) {

Paper otherPaper = (Paper) other;

**return** **this**.content.compareTo(otherPaper.content);

}

**return** 0;

}

}

**package** KI305.Panasiuk.Lab7;

/\*\*

\* A class representing a plastic bottle that implements the Item interface.

\* It provides methods to get the volume of the bottle and to print its details.

\* This class also implements the Comparable interface for comparison with other items.

\*

\* **@author** Taras Panasiuk

\* **@version** 1.0

\*/

**public** **class** PlasticBottle **implements** Item {

**private** **int** volume;

/\*\*

\* Constructs a plastic bottle with the specified volume.

\*

\* **@param** volume The volume of the plastic bottle in milliliters.

\*/

**public** PlasticBottle(**int** volume) {

**this**.volume = volume;

}

/\*\*

\* Gets the volume of the plastic bottle.

\*

\* **@return** The volume of the plastic bottle in milliliters.

\*/

**public** **int** getVolume() {

**return** volume;

}

@Override

**public** **int** getSize() {

**return** volume;

}

@Override

**public** **void** print() {

System.***out***.println("Plastic Bottle, Volume: " + volume + " mL");

}

@Override

**public** **int** compareTo(Item other) {

**if** (other **instanceof** PlasticBottle) {

PlasticBottle otherBottle = (PlasticBottle) other;

**return** Integer.*compare*(**this**.volume, otherBottle.volume);

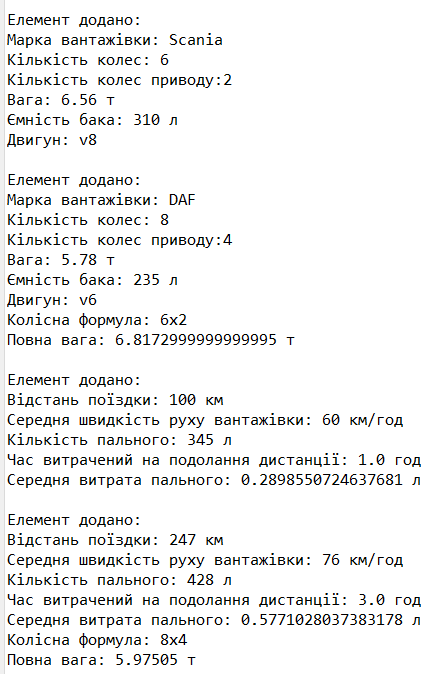
}

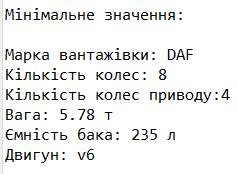
**return** 0;

}

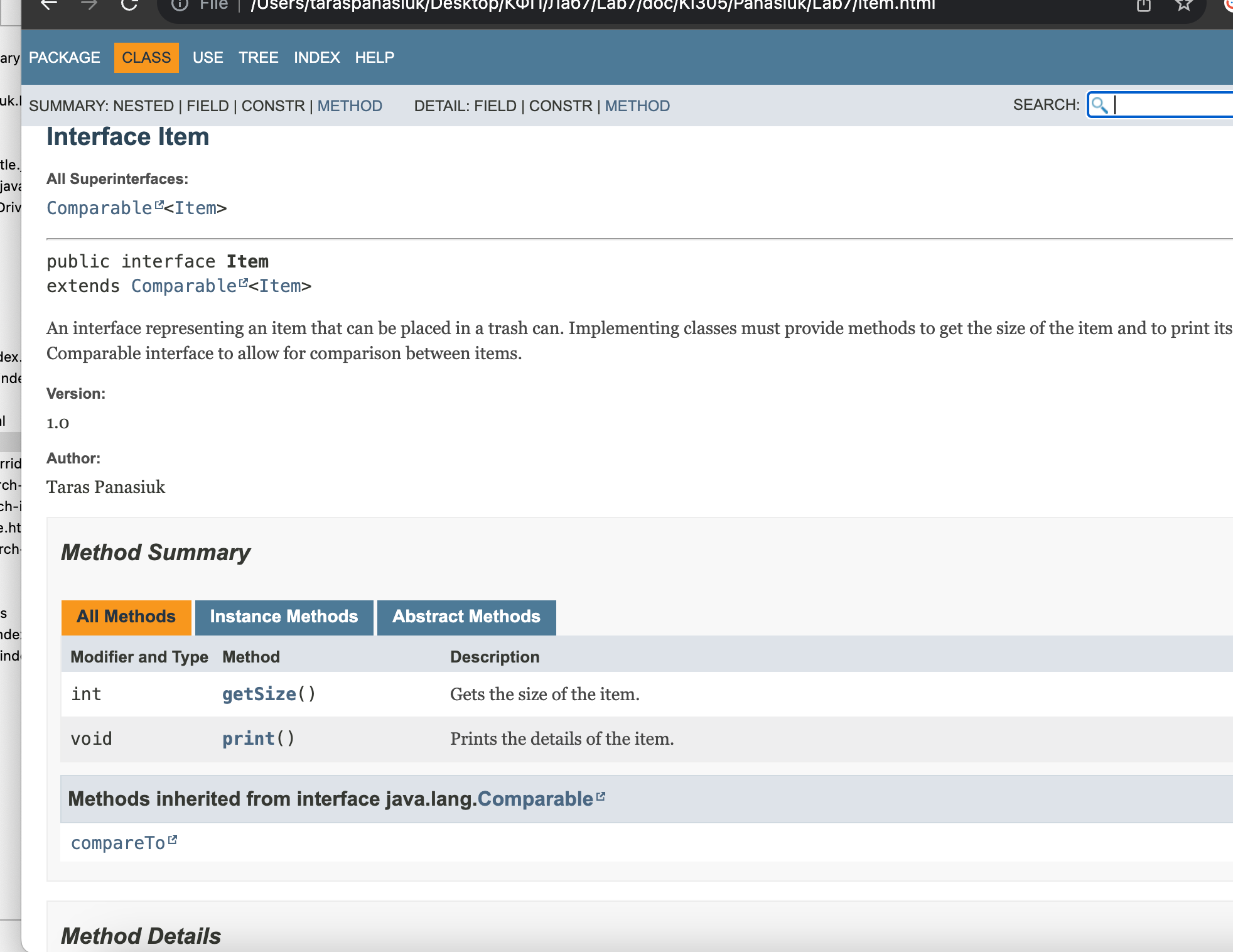
}

**Результат роботи програми у консолі та файлі:**

****

****

**Фрагмент генерації документації:**



**КОНТРОЛЬНІ ПИТАННЯ**

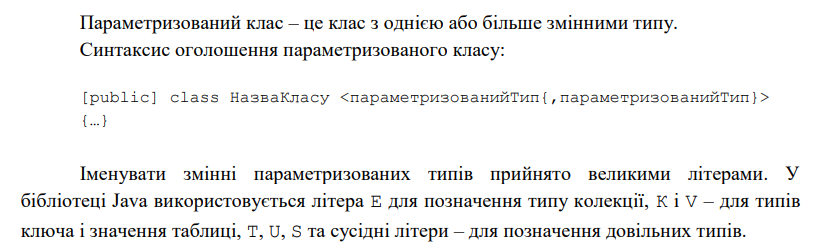
1. Дайте визначення терміну «параметризоване програмування».

**Відповідь:**

Параметризоване програмування є аналогом шаблонів у С++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів. Користувачів параметризованого програмування можна поділити на 3 рівні кваліфікації: 1. ті, що користуються готовими класами; 2. ті, що користуються готовими класами і вміють виправляти помилки, що виникають при їх використанні; 3. ті, що пишуть власні параметризовані класи.

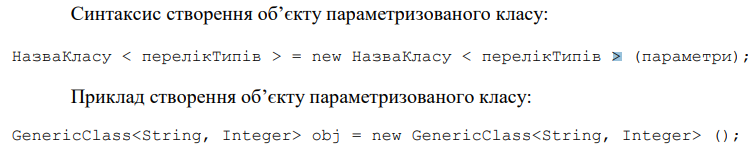
1. Розкрийте синтаксис визначення простого параметризованого класу.

**Відповідь:**

****

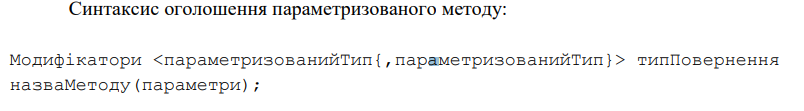
1. Розкрийте синтаксис створення об’єкту параметризованого класу.

**Відповідь:**

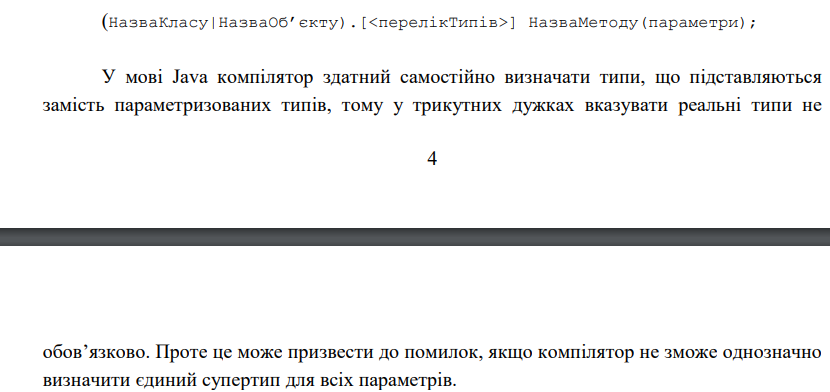
****

1. Розкрийте синтаксис визначення параметризованого методу.

**Відповідь:**

****

1. Розкрийте синтаксис виклику параметризованого методу.**Відповідь:**

****

1. Яку роль відіграє встановлення обмежень для змінних типів?

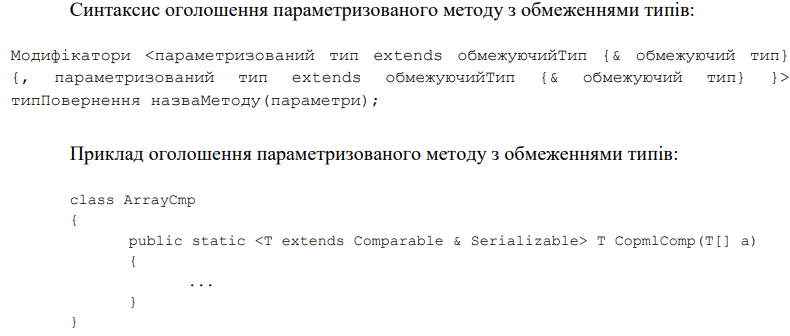
**Відповідь:**

Встановлення обмежень для змінних типів грає важливу роль у програмуванні з декількох причин:

1. Безпека даних: Обмеження можуть забезпечити захист від некоректного використання даних та неправильного доступу до них. Встановлення обмежень може допомогти уникнути помилок, пов'язаних з некоректними значеннями змінних.
2. Ефективність та оптимізація: Встановлення обмежень може допомогти оптимізувати використання пам'яті та обчислювальних ресурсів. Наприклад, використання коротших типів даних (наприклад, byte замість int) для зберігання значень, які можуть бути в межах діапазону byte, може зменшити обсяг пам'яті, потрібний для зберігання цих даних.
3. Читабельність та розуміння коду: Обмеження можуть слугувати як документація для програмістів, які використовують ваш код, надаючи інформацію про очікувані значення та діапазони, що можуть бути використані для конкретних змінних.
4. Підвищення стабільності та надійності програм: Перевірка обмежень може допомогти уникнути виникнення помилок, таких як переповнення пам'яті, витік пам'яті або інші види помилок, пов'язаних з некоректними значеннями змінних.
5. Як встановити обмеження для змінних типів?

**Відповідь:**

Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово extends і вказати один суперклас, або довільну кількість інтерфейсів (через знак &), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова extends.



1. Розкрийте правила спадкування параметризованих типів.

**Відповідь:**

У Java параметризовані типи (generics) дозволяють створювати класи та методи, які можна параметризувати типами даних. Коли ми працюємо зі спадкуванням параметризованих типів, виникають певні правила:

1. Підтримка підтипів: Параметризовані типи підтримують підтипи. Це означає, що якщо B є підтипом A, то MyClass<B> є підтипом MyClass<A>.
2. Не можна створювати інстанції типу з параметром: Не можна створити новий об'єкт параметризованого типу напряму, наприклад, new T() або new MyClass<T>(). Це обмеження пояснюється відсутністю відомостей про тип T на етапі компіляції.
3. Спільне використання методів і поля: При спадкуванні параметризованого класу або інтерфейсу всі методи і поля базового класу також доступні в похідних класах.
4. Передача параметризованих типів в конструктори: Підклас може передавати параметризовані типи до конструктора свого суперкласу.
5. Спеціалізація параметризованого типу: В деяких випадках можливе спеціалізування (розширення) параметризованого типу. Наприклад, MyClass<T> може бути розширено класом MyClassSpecialized<T, U>.
6. Перевірка типів за під час компіляції: Параметризовані типи перевіряються за типом під час компіляції, що забезпечує безпеку типів.

ризованих типів.

1. Яке призначення підстановочних типів?

**Відповідь:**

Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів.

1. Застосування підстановочних типів.

**Відповідь:**

Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;

2. обмеження супертипу;

3. необмежені підстановки.

**Висновок:** на цій лабораторній роботі я оволодів навиками параметризованого програмування мовою Java. Створив параметризований клас, який працює з двома класами, екземпляри якого розміщуються у списку цього класу в програмі-драйвері.