ЛАБОРАТОРНА РОБОТА № 3

Використання узагальнень (generics). Клонування та порівняння об'єктів.

Мета: створити міні проект Game з використанням узагальнень, клонування та порівняння об'єктів.

Хід роботи:

Завдання 1. Відкрити заготовлений проект з реалізованою базовою функціональністю.

Завдання 2. За допомогою узагальнень (generics) встановити такі обмеження:
- до команди можна додавати тільки учасників, що відносяться до одної ліги (Schoolar, Student або Employee). - - грати між собою можуть тільки команди з учасниками одної ліги (тобто команда студентів може грати тільки іншою командою студентів). - продемонструвати створення команд, гравців, додавання гравців до команд, гри між ними.

Лістинг завдання:

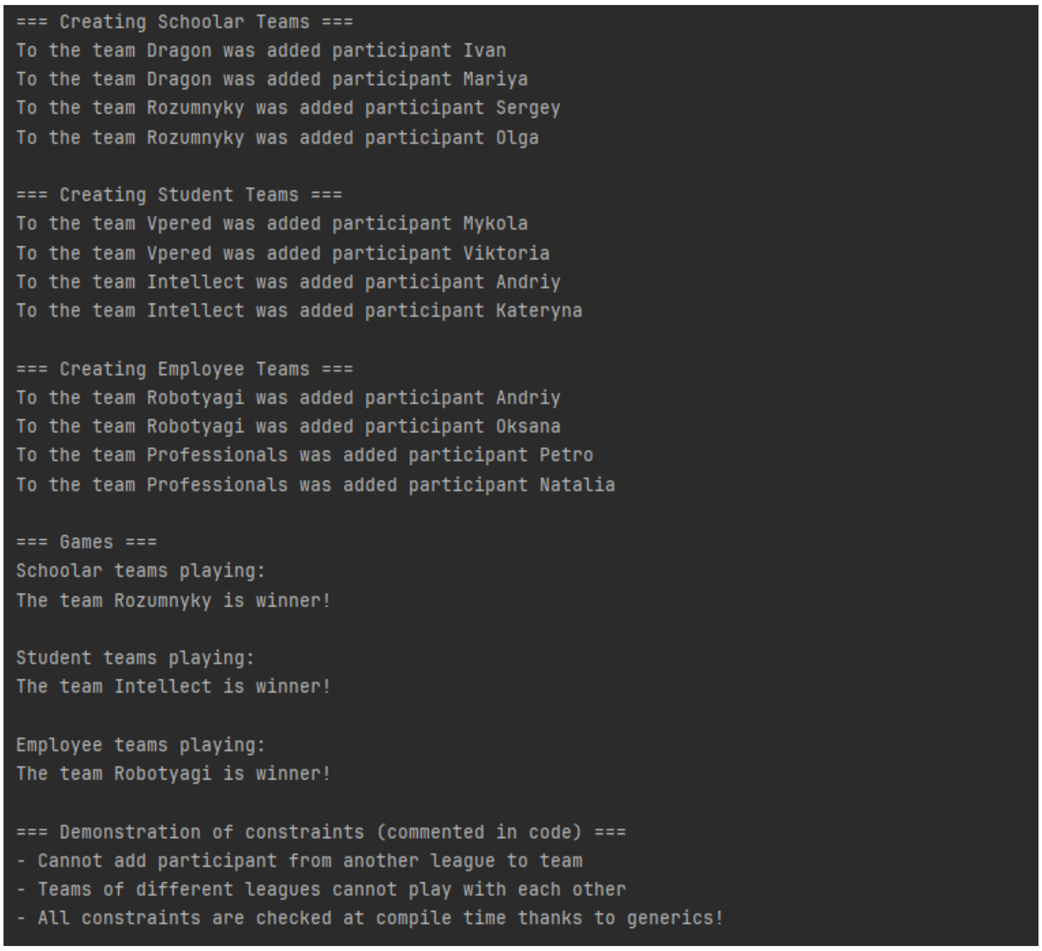
Team.java

```java
package com.education.ztu.game;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;

public class Team<T extends Participant> {
    private String name;
    private List<T> participants = new ArrayList<>();

    public Team(String name) {
        this.name = name;
    }

    public Team(Team<T> other) {
        this.name = other.name;
        this.participants = new ArrayList<>();
        for (T participant : other.participants) {
            this.participants.add((T) participant.clone());
        }
    }

    public static <T extends Participant> Team<T> deepClone(Team<T> original) {
        return new Team<>(original);
    }
    public void addNewParticipant(T participant) {
        participants.add(participant);
        System.out.println("To the team " + name + " was added participant " +
participant.getName());
    }

    public void playWith(Team<T> team) {
        String winnerName;
        Random random = new Random();
        int i = random.nextInt(2);
        if (i == 0) {
            winnerName = this.name;
        } else {
            winnerName = team.name;
        }
        System.out.println("The team " + winnerName + " is winner!");
    }

    public String getName() {
        return name;
    }

    public List<T> getParticipants() {
        return participants;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setParticipants(List<T> participants) {
        this.participants = participants;
    }

    @Override
    public String toString() {
        return "Team{" +
                "name='" + name + '\'' +
                ", participants=" + participants +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Team<?> team = (Team<?>) o;
        return Objects.equals(name, team.name) && Objects.equals(participants,
team.participants);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, participants);
    }
}
```

Game.java

```
package com.education.ztu.game;

public class Game {
    public static void main(String[] args) {
        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        Schoolar schoolar2 = new Schoolar("Mariya", 15);
        Schoolar schoolar3 = new Schoolar("Sergey", 12);
        Schoolar schoolar4 = new Schoolar("Olga", 14);

        Student student1 = new Student("Mykola", 20);
        Student student2 = new Student("Viktoria", 21);
        Student student3 = new Student("Andriy", 22);
        Student student4 = new Student("Kateryna", 19);

        Employee employee1 = new Employee("Andriy", 28);
        Employee employee2 = new Employee("Oksana", 25);
        Employee employee3 = new Employee("Petro", 30);
        Employee employee4 = new Employee("Natalia", 27);

        System.out.println("=== Creating Schoolar Teams ===");
        Team<Schoolar> schoolarTeam1 = new Team<>("Dragon");
        schoolarTeam1.addNewParticipant(schoolar1);
        schoolarTeam1.addNewParticipant(schoolar2);

        Team<Schoolar> schoolarTeam2 = new Team<>("Rozumnyky");
        schoolarTeam2.addNewParticipant(schoolar3);
        schoolarTeam2.addNewParticipant(schoolar4);

        System.out.println("\n=== Creating Student Teams ===");
        Team<Student> studentTeam1 = new Team<>("Vpered");
        studentTeam1.addNewParticipant(student1);
        studentTeam1.addNewParticipant(student2);

        Team<Student> studentTeam2 = new Team<>("Intellect");
        studentTeam2.addNewParticipant(student3);
        studentTeam2.addNewParticipant(student4);

        System.out.println("\n=== Creating Employee Teams ===");
        Team<Employee> employeeTeam1 = new Team<>("Robotyagi");
        employeeTeam1.addNewParticipant(employee1);
        employeeTeam1.addNewParticipant(employee2);

        Team<Employee> employeeTeam2 = new Team<>("Professionals");
        employeeTeam2.addNewParticipant(employee3);
        employeeTeam2.addNewParticipant(employee4);

        System.out.println("\n=== Games ===");
        System.out.println("Schoolar teams playing:");
        schoolarTeam1.playWith(schoolarTeam2);

        System.out.println("\nStudent teams playing:");
        studentTeam1.playWith(studentTeam2);

        System.out.println("\nEmployee teams playing:");
        employeeTeam1.playWith(employeeTeam2);

        System.out.println("\n=== Demonstration of constraints (commented in code)
===");
        System.out.println("- Cannot add participant from another league to
team");
        System.out.println("- Teams of different leagues cannot play with each
other");
        System.out.println("- All constraints are checked at compile time thanks
to generics!");
    }
}
```

Результат виконання:



```
=== Creating Schoolar Teams ===
To the team Dragon was added participant Ivan
To the team Dragon was added participant Mariya
To the team Rozumnyky was added participant Sergey
To the team Rozumnyky was added participant Olga

=== Creating Student Teams ===
To the team Vpered was added participant Mykola
To the team Vpered was added participant Viktoria
To the team Intellect was added participant Andriy
To the team Intellect was added participant Kateryna

=== Creating Employee Teams ===
To the team Robotyagi was added participant Andriy
To the team Robotyagi was added participant Oksana
To the team Professionals was added participant Petro
To the team Professionals was added participant Natalia

=== Games ===
Schoolar teams playing:
The team Rozumnyky is winner!

Student teams playing:
The team Intellect is winner!

Employee teams playing:
The team Robotyagi is winner!

=== Demonstration of constraints (commented in code) ===
- Cannot add participant from another league to team
- Teams of different leagues cannot play with each other
- All constraints are checked at compile time thanks to generics!
```

Рис. 1 Результат виконання завдання №2

Завдання 3. Клонування: - для класу Participant імплементувати інтерфейс Cloneable та перевизначити метод clone. - для класу Participant перевизначити методи hashCode та equals. - для класу Participant та його підкласів перевизначити метод toString. - для класу Team Реалізувати глибоке клонування через статичний метод або конструктор копіювання. - продемонструвати клонування та використання методів hashCode, equals та toString.

Лістинг завдання:

Schoolar.java

```java
package com.education.ztu.game;

public class Schoolar extends Participant {
    public Schoolar(String name, int age) {
        super(name, age);
    }

    @Override
    public String toString() {
        return "Schoolar{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                '}';
    }
}
```

Student.java

```java
package com.education.ztu.game;

public class Student extends Participant {
    public Student(String name, int age) {
        super(name, age);
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                '}';
    }
}
```

Employee.java

```java
package com.education.ztu.game;

public class Employee extends Participant {
    public Employee(String name, int age) {
        super(name, age);
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + getName() + '\'' +
                ", age=" + getAge() +
                '}';
    }
}
```

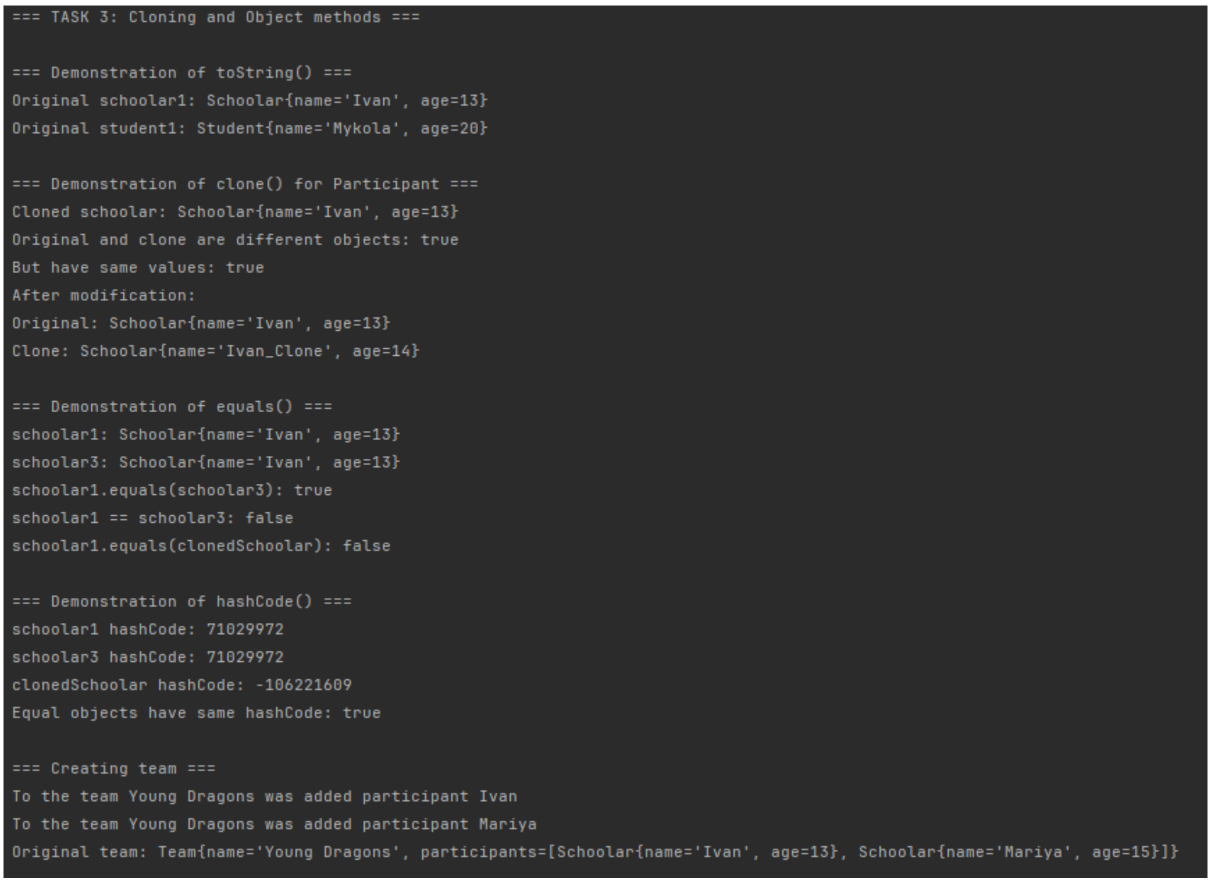
Team.java

```java
package com.education.ztu.game;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.Random;

public class Team<T extends Participant> {
    private String name;
    private List<T> participants = new ArrayList<>();

    public Team(String name) {
        this.name = name;
    }

    public Team(Team<T> other) {
        this.name = other.name;
        this.participants = new ArrayList<>();
        for (T participant : other.participants) {
            this.participants.add((T) participant.clone());
        }
    }
    public static <T extends Participant> Team<T> deepClone(Team<T> original) {
        return new Team<>(original);
    }

    public void addNewParticipant(T participant) {
        participants.add(participant);
        System.out.println("To the team " + name + " was added participant " +
participant.getName());
    }

    public void playWith(Team<T> team) {
        String winnerName;
        Random random = new Random();
        int i = random.nextInt(2);
        if (i == 0) {
            winnerName = this.name;
        } else {
            winnerName = team.name;
        }
        System.out.println("The team " + winnerName + " is winner!");
    }

    public String getName() {
        return name;
    }

    public List<T> getParticipants() {
        return participants;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setParticipants(List<T> participants) {
        this.participants = participants;
    }

    @Override
    public String toString() {
        return "Team{" +
                "name='" + name + '\'' +
                ", participants=" + participants +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Team<?> team = (Team<?>) o;
        return Objects.equals(name, team.name) && Objects.equals(participants,
team.participants);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, participants);
    }
}
```

Main.java

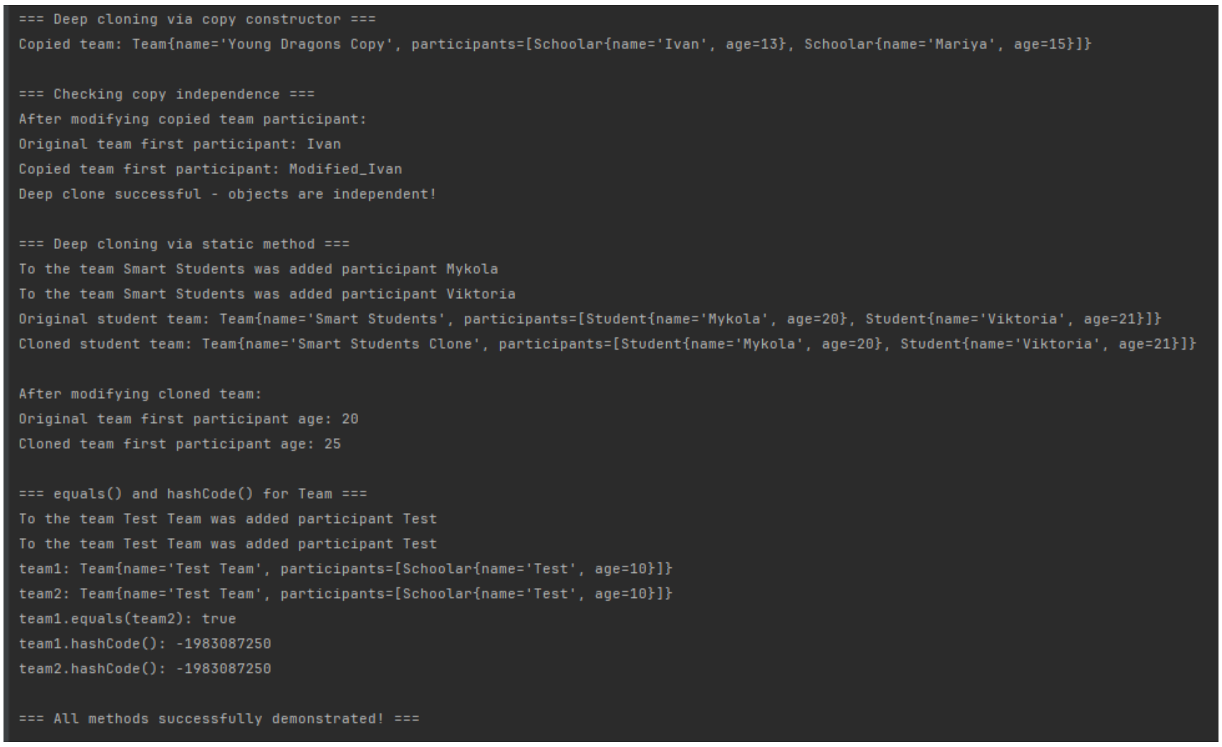
```java
package com.education.ztu;

import com.education.ztu.game.*;

public class Main {

    public static void main(String[] args) {
        System.out.println("=== TASK 3: Cloning and Object methods ===\n");

        Schoolar schoolar1 = new Schoolar("Ivan", 13);
        Schoolar schoolar2 = new Schoolar("Mariya", 15);
        Student student1 = new Student("Mykola", 20);

        System.out.println("=== Demonstration of toString() ===");
        System.out.println("Original schoolar1: " + schoolar1);
        System.out.println("Original student1: " + student1);

        System.out.println("\n=== Demonstration of clone() for Participant ===");
        Schoolar clonedSchoolar = (Schoolar) schoolar1.clone();
        System.out.println("Cloned schoolar: " + clonedSchoolar);
        System.out.println("Original and clone are different objects: " +
(schoolar1 != clonedSchoolar));
        System.out.println("But have same values: " +
schoolar1.equals(clonedSchoolar));

        clonedSchoolar.setName("Ivan_Clone");
        clonedSchoolar.setAge(14);
        System.out.println("After modification:");
        System.out.println("Original: " + schoolar1);
        System.out.println("Clone: " + clonedSchoolar);

        System.out.println("\n=== Demonstration of equals() ===");
        Schoolar schoolar3 = new Schoolar("Ivan", 13);
        System.out.println("schoolar1: " + schoolar1);
        System.out.println("schoolar3: " + schoolar3);
        System.out.println("schoolar1.equals(schoolar3): " +
schoolar1.equals(schoolar3));
        System.out.println("schoolar1 == schoolar3: " + (schoolar1 == schoolar3));
        System.out.println("schoolar1.equals(clonedSchoolar): " +
schoolar1.equals(clonedSchoolar));

        System.out.println("\n=== Demonstration of hashCode() ===");
        System.out.println("schoolar1 hashCode: " + schoolar1.hashCode());
        System.out.println("schoolar3 hashCode: " + schoolar3.hashCode());
        System.out.println("clonedSchoolar hashCode: " +
clonedSchoolar.hashCode());
        System.out.println("Equal objects have same hashCode: " +
                (schoolar1.equals(schoolar3) && schoolar1.hashCode() ==
schoolar3.hashCode()));

        System.out.println("\n=== Creating team ===");
        Team<Schoolar> originalTeam = new Team<>("Young Dragons");
        originalTeam.addNewParticipant(schoolar1);
        originalTeam.addNewParticipant(schoolar2);
        System.out.println("Original team: " + originalTeam);

        System.out.println("\n=== Deep cloning via copy constructor ===");
        Team<Schoolar> copiedTeam = new Team<>(originalTeam);
        copiedTeam.setName("Young Dragons Copy");
        System.out.println("Copied team: " + copiedTeam);
        System.out.println("\n=== Checking copy independence ===");
        copiedTeam.getParticipants().get(0).setName("Modified_Ivan");
        System.out.println("After modifying copied team participant:");
        System.out.println("Original team first participant: " +
                originalTeam.getParticipants().get(0).getName());
        System.out.println("Copied team first participant: " +
                copiedTeam.getParticipants().get(0).getName());
        System.out.println("Deep clone successful - objects are independent!");

        System.out.println("\n=== Deep cloning via static method ===");
        Team<Student> studentTeam = new Team<>("Smart Students");
        studentTeam.addNewParticipant(student1);
        studentTeam.addNewParticipant(new Student("Viktoria", 21));
        System.out.println("Original student team: " + studentTeam);

        Team<Student> clonedStudentTeam = Team.deepClone(studentTeam);
        clonedStudentTeam.setName("Smart Students Clone");
        System.out.println("Cloned student team: " + clonedStudentTeam);

        clonedStudentTeam.getParticipants().get(0).setAge(25);
        System.out.println("\nAfter modifying cloned team:");
        System.out.println("Original team first participant age: " +
                studentTeam.getParticipants().get(0).getAge());
        System.out.println("Cloned team first participant age: " +
                clonedStudentTeam.getParticipants().get(0).getAge());

        System.out.println("\n=== equals() and hashCode() for Team ===");
        Team<Schoolar> team1 = new Team<>("Test Team");
        team1.addNewParticipant(new Schoolar("Test", 10));

        Team<Schoolar> team2 = new Team<>("Test Team");
        team2.addNewParticipant(new Schoolar("Test", 10));

        System.out.println("team1: " + team1);
        System.out.println("team2: " + team2);
        System.out.println("team1.equals(team2): " + team1.equals(team2));
        System.out.println("team1.hashCode(): " + team1.hashCode());
        System.out.println("team2.hashCode(): " + team2.hashCode());

        System.out.println("\n=== All methods successfully demonstrated! ===");
    }
}
```

Результат виконання:



```
=== TASK 3: Cloning and Object methods ===

=== Demonstration of toString() ===
Original schoolar1: Schoolar{name='Ivan', age=13}
Original student1: Student{name='Mykola', age=20}

=== Demonstration of clone() for Participant ===
Cloned schoolar: Schoolar{name='Ivan', age=13}
Original and clone are different objects: true
But have same values: true
After modification:
Original: Schoolar{name='Ivan', age=13}
Clone: Schoolar{name='Ivan_Clone', age=14}

=== Demonstration of equals() ===
schoolar1: Schoolar{name='Ivan', age=13}
schoolar3: Schoolar{name='Ivan', age=13}
schoolar1.equals(schoolar3): true
schoolar1 == schoolar3: false
schoolar1.equals(clonedSchoolar): false

=== Demonstration of hashCode() ===
schoolar1 hashCode: 71029972
schoolar3 hashCode: 71029972
clonedSchoolar hashCode: -106221609
Equal objects have same hashCode: true

=== Creating team ===
To the team Young Dragons was added participant Ivan
To the team Young Dragons was added participant Mariya
Original team: Team{name='Young Dragons', participants=[Schoolar{name='Ivan', age=13}, Schoolar{name='Mariya', age=15}]}
```

Рис. 2 Виконна завдання 3 частина №1.



```
=== Deep cloning via copy constructor ===
Copied team: Team{name='Young Dragons Copy', participants=[Schoolar{name='Ivan', age=13}, Schoolar{name='Mariya', age=15}]}

=== Checking copy independence ===
After modifying copied team participant:
Original team first participant: Ivan
Copied team first participant: Modified_Ivan
Deep clone successful - objects are independent!

=== Deep cloning via static method ===
To the team Smart Students was added participant Mykola
To the team Smart Students was added participant Viktoria
Original student team: Team{name='Smart Students', participants=[Student{name='Mykola', age=20}, Student{name='Viktoria', age=21}]}
Cloned student team: Team{name='Smart Students Clone', participants=[Student{name='Mykola', age=20}, Student{name='Viktoria', age=21}]}

After modifying cloned team:
Original team first participant age: 20
Cloned team first participant age: 25

=== equals() and hashCode() for Team ===
To the team Test Team was added participant Test
To the team Test Team was added participant Test
team1: Team{name='Test Team', participants=[Schoolar{name='Test', age=10}]}
team2: Team{name='Test Team', participants=[Schoolar{name='Test', age=10}]}
team1.equals(team2): true
team1.hashCode(): -1983087250
team2.hashCode(): -1983087250

=== All methods successfully demonstrated! ===
```

Рис. 3 Виконна завдання 3 частина №2.

Завдання 4. Порівняння: - для класу Participant імплементувати інтерфейс Comparable та перевизначити метод compareTo для сортування учасників по імені.

- створити Comparator для порівняння учасників по віку. - *створити компаратор з пріорітетом використовуючи можливості Java 8 (спочатку порівняння по імені, а потім по віку). - продемонструвати роботу порівнянь на прикладі сортування учасників команд.

Лістинг завдання:

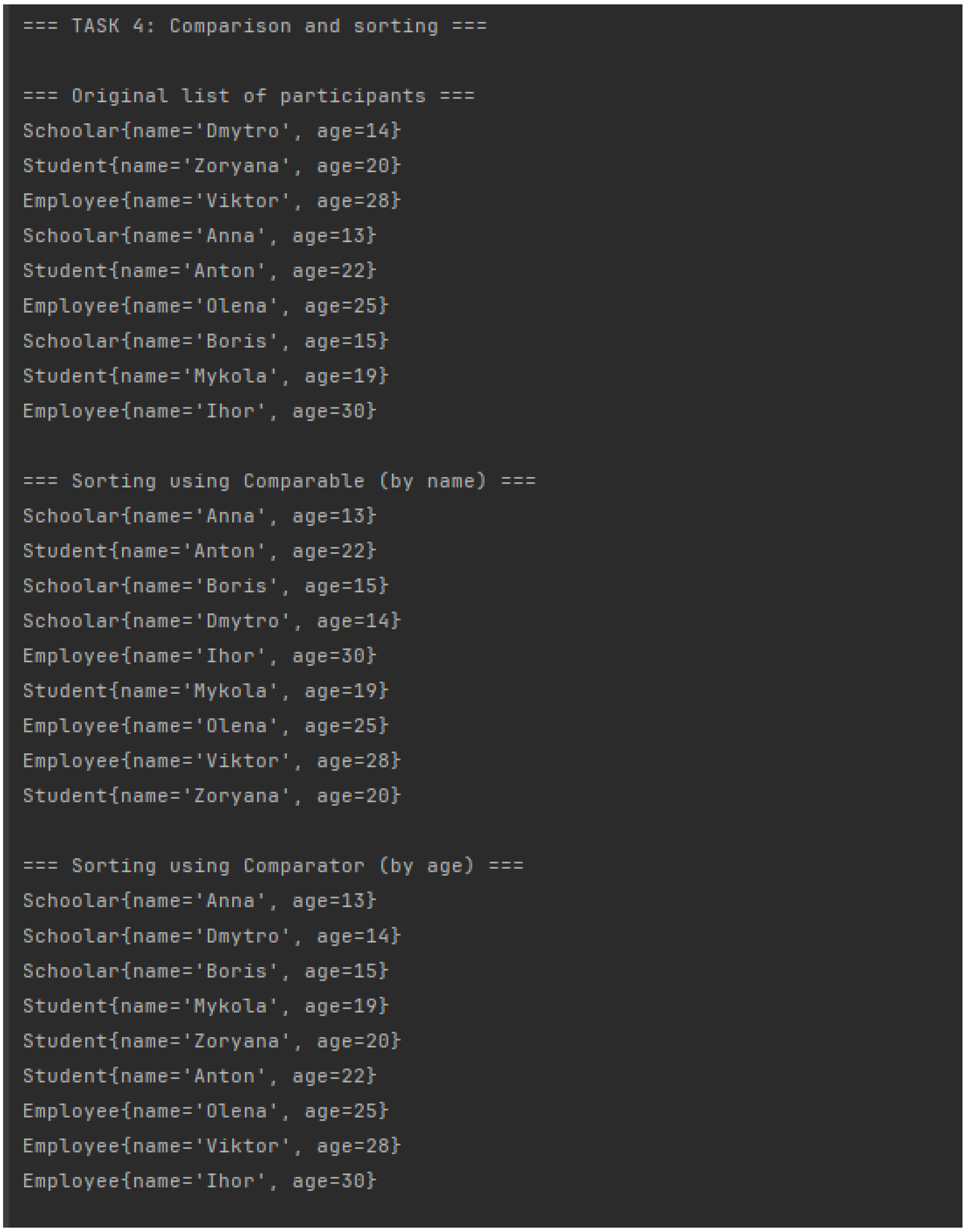
Participant.java

```
package com.education.ztu.game;

import java.util.Objects;

public abstract class Participant implements Cloneable, Comparable<Participant> {
    private String name;
    private int age;

    public Participant(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public Participant clone() {
        try {
            return (Participant) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException("Clone not supported", e);
        }
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Participant that = (Participant) o;
        return age == that.age && Objects.equals(name, that.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
    @Override
    public String toString() {
        return "Participant{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Participant other) {
        return this.name.compareTo(other.name);
    }
}
```

Main.java

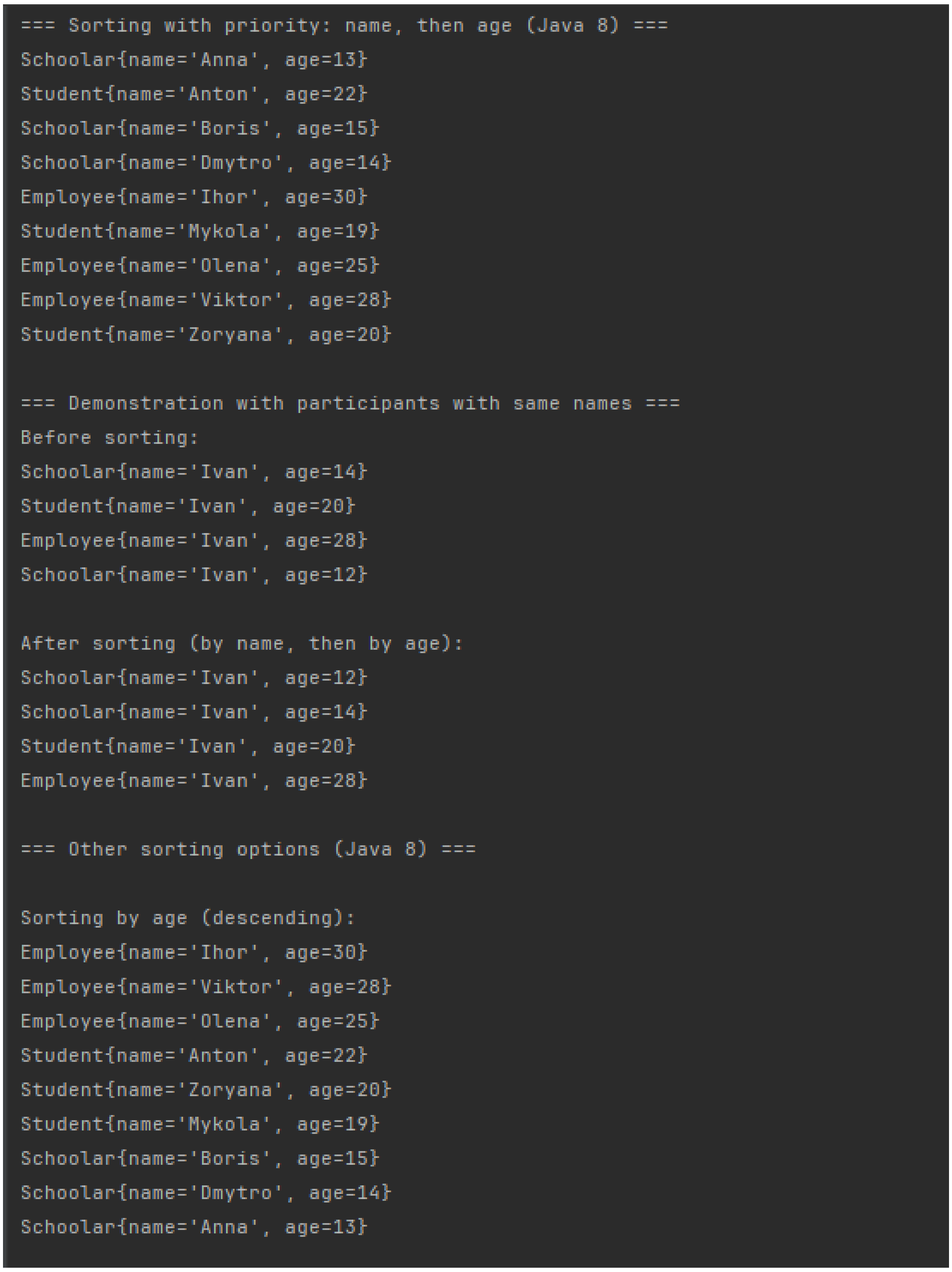
```
package com.education.ztu;

import com.education.ztu.game.*;
import java.util.*;

public class Main2 {
    public static void main(String[] args) {
        System.out.println("=== TASK 4: Comparison and sorting ===\n");

        Team<Participant> team = new Team<>("Mixed Team");

        Schoolar schoolar1 = new Schoolar("Dmytro", 14);
        Schoolar schoolar2 = new Schoolar("Anna", 13);
        Schoolar schoolar3 = new Schoolar("Boris", 15);

        Student student1 = new Student("Zoryana", 20);
        Student student2 = new Student("Anton", 22);
        Student student3 = new Student("Mykola", 19);

        Employee employee1 = new Employee("Viktor", 28);
        Employee employee2 = new Employee("Olena", 25);
        Employee employee3 = new Employee("Ihor", 30);

        List<Participant> participants = new ArrayList<>();
        participants.add(schoolar1);
        participants.add(student1);
        participants.add(employee1);
        participants.add(schoolar2);
        participants.add(student2);
        participants.add(employee2);
        participants.add(schoolar3);
        participants.add(student3);
        participants.add(employee3);

        System.out.println("=== Original list of participants ===");
        for (Participant p : participants) {
            System.out.println(p);
        }

        System.out.println("\n=== Sorting using Comparable (by name) ===");
        List<Participant> sortedByName = new ArrayList<>(participants);
        Collections.sort(sortedByName);
        for (Participant p : sortedByName) {
            System.out.println(p);
        }

        System.out.println("\n=== Sorting using Comparator (by age) ===");
        Comparator<Participant> ageComparator = new Comparator<Participant>() {
            @Override
            public int compare(Participant p1, Participant p2) {
                return Integer.compare(p1.getAge(), p2.getAge());
            }
        };

        List<Participant> sortedByAge = new ArrayList<>(participants);
        Collections.sort(sortedByAge, ageComparator);
        for (Participant p : sortedByAge) {
            System.out.println(p);
        }

        System.out.println("\n=== Sorting with priority: name, then age (Java 8)
===");
        Comparator<Participant> nameAndAgeComparator = Comparator
                .comparing(Participant::getName)
                .thenComparing(Participant::getAge);

        List<Participant> sortedByNameAndAge = new ArrayList<>(participants);
        Collections.sort(sortedByNameAndAge, nameAndAgeComparator);
        for (Participant p : sortedByNameAndAge) {
            System.out.println(p);
        }

        System.out.println("\n=== Demonstration with participants with same names
===");
        List<Participant> sameNameParticipants = new ArrayList<>();
        sameNameParticipants.add(new Schoolar("Ivan", 14));
        sameNameParticipants.add(new Student("Ivan", 20));
        sameNameParticipants.add(new Employee("Ivan", 28));
        sameNameParticipants.add(new Schoolar("Ivan", 12));

        System.out.println("Before sorting:");
        for (Participant p : sameNameParticipants) {
            System.out.println(p);
        }

        Collections.sort(sameNameParticipants, nameAndAgeComparator);
        System.out.println("\nAfter sorting (by name, then by age):");
        for (Participant p : sameNameParticipants) {
            System.out.println(p);
        }

        System.out.println("\n=== Other sorting options (Java 8) ===");

        System.out.println("\nSorting by age (descending):");
        List<Participant> sortedByAgeDesc = new ArrayList<>(participants);

sortedByAgeDesc.sort(Comparator.comparing(Participant::getAge).reversed());
        for (Participant p : sortedByAgeDesc) {
            System.out.println(p);
        }

        System.out.println("\nSorting by name (reversed), then by age:");
        List<Participant> sortedComplex = new ArrayList<>(participants);
        sortedComplex.sort(Comparator.comparing(Participant::getName).reversed()
                .thenComparing(Participant::getAge));
        for (Participant p : sortedComplex) {
            System.out.println(p);
        }

        System.out.println("\n=== Using Stream API for sorting ===");
        System.out.println("Top 5 youngest participants:");
        participants.stream()
                .sorted(Comparator.comparing(Participant::getAge))
                .limit(5)
                .forEach(System.out::println);

        System.out.println("\n=== All comparison methods successfully 
demonstrated! ===");
    }
}
```

Результат виконання:



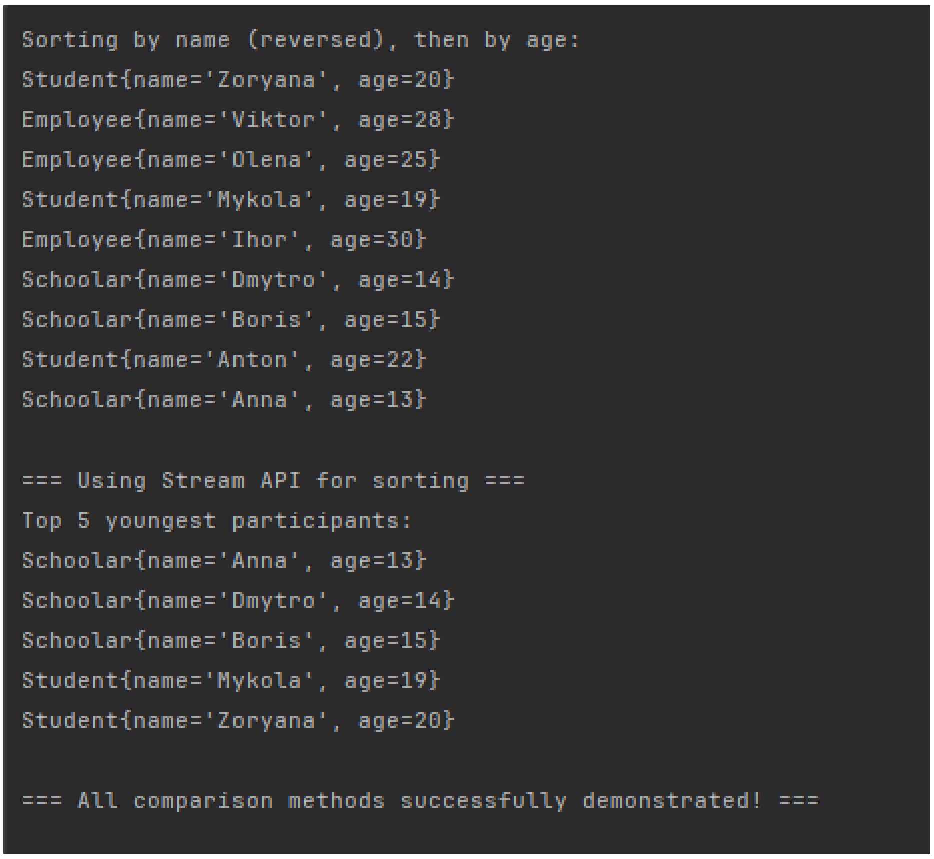
```
=== TASK 4: Comparison and sorting ===

=== Original list of participants ===
Schoolar{name='Dmytro', age=14}
Student{name='Zoryana', age=20}
Employee{name='Viktor', age=28}
Schoolar{name='Anna', age=13}
Student{name='Anton', age=22}
Employee{name='Olena', age=25}
Schoolar{name='Boris', age=15}
Student{name='Mykola', age=19}
Employee{name='Ihor', age=30}

=== Sorting using Comparable (by name) ===
Schoolar{name='Anna', age=13}
Student{name='Anton', age=22}
Schoolar{name='Boris', age=15}
Schoolar{name='Dmytro', age=14}
Employee{name='Ihor', age=30}
Student{name='Mykola', age=19}
Employee{name='Olena', age=25}
Employee{name='Viktor', age=28}
Student{name='Zoryana', age=20}

=== Sorting using Comparator (by age) ===
Schoolar{name='Anna', age=13}
Schoolar{name='Dmytro', age=14}
Schoolar{name='Boris', age=15}
Student{name='Mykola', age=19}
Student{name='Zoryana', age=20}
Student{name='Anton', age=22}
Employee{name='Olena', age=25}
Employee{name='Viktor', age=28}
Employee{name='Ihor', age=30}
```

Рис. 4 Виконна завдання 4 частина №1.



```
=== Sorting with priority: name, then age (Java 8) ===
Schoolar{name='Anna', age=13}
Student{name='Anton', age=22}
Schoolar{name='Boris', age=15}
Schoolar{name='Dmytro', age=14}
Employee{name='Ihor', age=30}
Student{name='Mykola', age=19}
Employee{name='Olena', age=25}
Employee{name='Viktor', age=28}
Student{name='Zoryana', age=20}

=== Demonstration with participants with same names ===
Before sorting:
Schoolar{name='Ivan', age=14}
Student{name='Ivan', age=20}
Employee{name='Ivan', age=28}
Schoolar{name='Ivan', age=12}

After sorting (by name, then by age):
Schoolar{name='Ivan', age=12}
Schoolar{name='Ivan', age=14}
Student{name='Ivan', age=20}
Employee{name='Ivan', age=28}

=== Other sorting options (Java 8) ===

Sorting by age (descending):
Employee{name='Ihor', age=30}
Employee{name='Viktor', age=28}
Employee{name='Olena', age=25}
Student{name='Anton', age=22}
Student{name='Zoryana', age=20}
Student{name='Mykola', age=19}
Schoolar{name='Boris', age=15}
Schoolar{name='Dmytro', age=14}
Schoolar{name='Anna', age=13}
```

Рис. 5 Виконна завдання 4 частина №2.



```
Sorting by name (reversed), then by age:
Student{name='Zoryana', age=20}
Employee{name='Viktor', age=28}
Employee{name='Olena', age=25}
Student{name='Mykola', age=19}
Employee{name='Ihor', age=30}
Schoolar{name='Dmytro', age=14}
Schoolar{name='Boris', age=15}
Student{name='Anton', age=22}
Schoolar{name='Anna', age=13}

=== Using Stream API for sorting ===
Top 5 youngest participants:
Schoolar{name='Anna', age=13}
Schoolar{name='Dmytro', age=14}
Schoolar{name='Boris', age=15}
Student{name='Mykola', age=19}
Student{name='Zoryana', age=20}

=== All comparison methods successfully demonstrated! ===
```

Рис. 6 Виконна завдання 4 частина №3.

Посилання на репозиторій: https://github.com/Oleg-Ischuk/Java-IPZ-23-1

Висновок: створив міні проект Game з використанням узагальнень, клонування та порівняння об'єктів.