OOP - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        Point2D point2D = new Point2D(10, 20);
        point2D.setXY(43, 65);
        System.out.println(point2D);
        Point3D point3D = new Point3D(10, 20, 30);
        point3D.setXYZ(43, 64, 2);
        System.out.println(point3D);
    }
}

class Point2D {
    protected float x, y;

    public Point2D() {

    }

    public Point2D(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }

    public void setY(float y) {
        this.y = y;
```

```
    }

    public float[] getXY() {
        return new float[]{x, y};
    }

    public void setXY(float x, float y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return String.format("(%f,%f)", x, y);
    }
}

class Point3D extends Point2D {
    private float z;

    public Point3D(float x, float y, float z) {
        super(x, y);
        this.z = z;
    }

    public float getZ() {
        return z;
    }

    public void setZ(float z) {
        this.z = z;
    }

    public float[] getXYZ() {
        return new float[]{x, y, z};
    }

    public void setXYZ(float x, float y, float z) {
        this.setXY(x, y);
        this.z = z;
    }

    @Override
    public String toString() {
        return String.format("(%f,%f,%f)", x, y, z);
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) {
        Person student = new Student("John", "BC 43", "IT", 1,
1000);
        Person staff = new Staff("Computer Programming", 4500f);
        System.out.println(student);
        System.out.println(staff);
    }
}

class Person {
    protected String name, address;

    public Person() {

    }

    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return String.format("%s->%s", name, address);
    }
}
```

```
class Student extends Person {
    private String typeOfStudies;
    private int yearOfStudy;
    private float studiesPrice;

    public Student(String name, String address, String
typeOfStudies, int yearOfStudy, float studiesPrice) {
        super(name, address);
        this.typeOfStudies = typeOfStudies;
        this.yearOfStudy = yearOfStudy;
        this.studiesPrice = studiesPrice;
    }

    public String getTypeOfStudies() {
        return typeOfStudies;
    }

    public void setTypeOfStudies(String typeOfStudies) {
        this.typeOfStudies = typeOfStudies;
    }

    public int getYearOfStudy() {
        return yearOfStudy;
    }

    public void setYearOfStudy(int yearOfStudy) {
        this.yearOfStudy = yearOfStudy;
    }

    public float getStudiesPrice() {
        return studiesPrice;
    }

    public void setStudiesPrice(float studiesPrice) {
        this.studiesPrice = studiesPrice;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                ", typeOfStudies='" + typeOfStudies + '\'' +
                ", yearOfStudy='" + yearOfStudy + '\'' +
                ", studiesPrice=" + studiesPrice +
                '}';
    }
}
```

```
class Staff extends Person {
    private String specialization;
    private float salary;

    public Staff(String specialization, float salary) {
        this.specialization = specialization;
        this.salary = salary;
    }

    public String getSpecialization() {
        return specialization;
    }

    public void setSpecialization(String specialization) {
        this.specialization = specialization;
    }

    public float getSalary() {
        return salary;
    }

    public void setSalary(float salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Staff{" +
                "name='" + name + '\'' +
                ", address='" + address + '\'' +
                ", specialization='" + specialization + '\'' +
                ", salary='" + salary + '\'' +
                '}';
    }
}
```

Task 3

```
public class Exercise3 {

    public static void main(String[] args) {
        Shape shape = new Shape("red", false);
        System.out.println(shape);

        Shape circle = new Circle("blue", true, 20);
        System.out.println(circle);
```

```
        Shape rectangle = new Rectangle("yellow", true, 20, 30);
        System.out.println(rectangle);

        Shape square = new Square("green", false, 40);
        System.out.println(square);
    }
}

class Shape {
    private String color;
    private boolean isFilled;

    public Shape() {
        this.color = "unknown";
        this.isFilled = false;
    }

    public Shape(String color, boolean isFilled) {
        this.color = color;
        this.isFilled = isFilled;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return isFilled;
    }

    public void setFilled(boolean filled) {
        isFilled = filled;
    }

    @Override
    public String toString() {
        return String.format("Shape with color of %s and %s",
color, isFilled ? "filled" : "NotFilled");
    }
}

class Circle extends Shape {
    private float radius;
```

```
    public Circle(String color, boolean isFilled, float radius) {
        super(color, isFilled);
        this.radius = radius;
    }

    public float getRadius() {
        return radius;
    }

    public void setRadius(float radius) {
        this.radius = radius;
    }

    public float getArea() {
        return (float) (Math.PI * radius * radius);
    }

    public float getPerimeter() {
        return (float) (2 * Math.PI * radius);
    }

    @Override
    public String toString() {
        return String.format("Circle with radius=%f which is
subclass off %s", radius, super.toString());
    }
}

class Rectangle extends Shape {
    protected double width, length;

    public Rectangle(String color, boolean isFilled, double width,
double length) {
        super(color, isFilled);
        this.width = width;
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }
```

```
    public void setLength(double length) {
        this.length = length;
    }

    public float getArea() {
        return (float) (width * length);
    }

    public float getPerimeter() {
        return (float) (2 * width + 2 * length);
    }

    @Override
    public String toString() {
        return String.format("Rectangle with width=%f and
length=%f which is subclass off %s", width, length,
super.toString());
    }
}

class Square extends Rectangle {

    public Square(String color, boolean isFilled, double size) {
        super(color, isFilled, size, size);
    }

    @Override
    public void setWidth(double width) {
        super.setWidth(width);
        super.setLength(width);
    }

    @Override
    public void setLength(double length) {
        super.setWidth(width);
        super.setLength(length);
    }

    @Override
    public String toString() {
        return String.format("Square with width=%f and length=%f
which is subclass off %s", width, length, super.toString());
    }
}
```

Task 4

```
public class Exercise4 {

    public static void main(String[] args) {
        ShapeEx4[] shapes = {
                new CircleEx4("blue", true, 20),
                new RectangleEx4("yellow", true, 20, 30),
                new SquareEx4("green", false, 40)
        };
        for (ShapeEx4 shape : shapes) {
            System.out.println(shape);
            System.out.println(shape.getArea());
            System.out.println(shape.getPerimeter());
        }
    }
}

abstract class ShapeEx4 {
    private String color;
    private boolean isFilled;

    public ShapeEx4() {
        this.color = "unknown";
        this.isFilled = false;
    }

    public ShapeEx4(String color, boolean isFilled) {
        this.color = color;
        this.isFilled = isFilled;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public boolean isFilled() {
        return isFilled;
    }

    public void setFilled(boolean filled) {
        isFilled = filled;
    }
```

```
    public abstract float getArea();

    public abstract float getPerimeter();

    @Override
    public String toString() {
        return String.format("Shape with color of %s and %s",
color, isFilled ? "filled" : "NotFilled");
    }
}

class CircleEx4 extends ShapeEx4 {
    private float radius;

    public CircleEx4(String color, boolean isFilled, float radius)
{
        super(color, isFilled);
        this.radius = radius;
    }

    public float getRadius() {
        return radius;
    }

    public void setRadius(float radius) {
        this.radius = radius;
    }

    @Override
    public float getArea() {
        return (float) (Math.PI * radius * radius);
    }

    @Override
    public float getPerimeter() {
        return (float) (2 * Math.PI * radius);
    }

    @Override
    public String toString() {
        return String.format("Circle with radius=%f which is
subclass off %s", radius, super.toString());
    }
}

class RectangleEx4 extends ShapeEx4 {
    protected double width, length;
```

```
    public RectangleEx4(String color, boolean isFilled, double
width, double length) {
        super(color, isFilled);
        this.width = width;
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        this.length = length;
    }

    @Override
    public float getArea() {
        return (float) (width * length);
    }

    @Override
    public float getPerimeter() {
        return (float) (2 * width + 2 * length);
    }

    @Override
    public String toString() {
        return String.format("Rectangle with width=%f and
length=%f which is subclass off %s", width, length,
super.toString());
    }
}

class SquareEx4 extends RectangleEx4 {

    public SquareEx4(String color, boolean isFilled, double size)
{
        super(color, isFilled, size, size);
    }

    @Override
```

```
    public void setWidth(double width) {
        super.setWidth(width);
        super.setLength(width);
    }

    @Override
    public void setLength(double length) {
        super.setWidth(width);
        super.setLength(length);
    }

    @Override
    public String toString() {
        return String.format("Square with width=%f and length=%f
which is subclass off %s", width, length, super.toString());
    }
}
```

Task 5

```
public class Exercise5 {

    public static void main(String[] args) {
        Line line = new Line(10, 20, 30, 40);
        System.out.println(line.getLength());
        System.out.println(line.getMiddlePoint());
    }
}

class Line {

    private Point2DExt p1, p2;

    public Line(Point2DExt p1, Point2DExt p2) {
        this.p1 = p1;
        this.p2 = p2;
    }

    public Line(float p1Start, float p1End, float p2Start, float
p2End) {
        this.p1 = new Point2DExt(p1Start, p1End);
        this.p2 = new Point2DExt(p2Start, p2End);
    }

    public Point2DExt getP1() {
        return p1;
```

```
    }

    public void setP1(Point2DExt p1) {
        this.p1 = p1;
    }

    public Point2DExt getP2() {
        return p2;
    }

    public void setP2(Point2DExt p2) {
        this.p2 = p2;
    }

    public float getLength() {
        return (float) Math.sqrt((Math.pow(p2.x - p1.x, 2) +
Math.pow(p2.y - p1.y, 2)));
    }

    public Point2DExt getMiddlePoint() {
        float xMiddle = (p1.x + p2.x) / 2;
        float yMiddle = (p1.y + p2.y) / 2;
        return new Point2DExt(xMiddle, yMiddle);
    }
}

class Point2DExt {
    protected float x, y;

    public Point2DExt() {

    }

    public Point2DExt(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }
```

```
    public void setY(float y) {
        this.y = y;
    }

    public float[] getXY() {
        return new float[]{x, y};
    }

    public void setXY(float x, float y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return String.format("(%f,%f)", x, y);
    }
}
```

Task 6

```
public class Exercise6 {

    public static void main(String[] args) {
        Movable movablePoint = new MovablePoint(10, 10, 3, 3);
        System.out.println(movablePoint);
        movablePoint.moveRight();
        movablePoint.moveUp();
        System.out.println(movablePoint);

        Movable movableCircle = new MovablePoint(20, 30, 4, 4);
        System.out.println(movableCircle);
        movableCircle.moveRight();
        movableCircle.moveUp();
        System.out.println(movableCircle);
    }
}

interface Movable {
    void moveUp();

    void moveDown();

    void moveLeft();
```

```
    void moveRight();
}

class MovablePoint implements Movable {

    private int x, y;
    private int xSpeed, ySpeed;

    public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
        this.x = x;
        this.y = y;
        this.xSpeed = xSpeed;
        this.ySpeed = ySpeed;
    }

    @Override
    public void moveUp() {
        y += ySpeed;
    }

    @Override
    public void moveDown() {
        y -= ySpeed;
    }

    @Override
    public void moveLeft() {
        x -= xSpeed;
    }

    @Override
    public void moveRight() {
        x += xSpeed;
    }

    @Override
    public String toString() {
        return "MovablePoint{" +
                "x=" + x +
                ", y=" + y +
                ", xSpeed=" + xSpeed +
                ", ySpeed=" + ySpeed +
                '}';
    }
}

class MovableCircle implements Movable {

    private float radius;
```

```
    private MovablePoint movablePoint;

    public MovableCircle(int radius, int x, int y, int xSpeed, int
ySpeed) {
        this.radius = radius;
        this.movablePoint = new MovablePoint(x, y, xSpeed,
ySpeed);
    }

    @Override
    public void moveUp() {
        movablePoint.moveUp();
    }

    @Override
    public void moveDown() {
        movablePoint.moveDown();
    }

    @Override
    public void moveLeft() {
        movablePoint.moveLeft();
    }

    @Override
    public void moveRight() {
        movablePoint.moveRight();
    }

    @Override
    public String toString() {
        return "MovableCircle{" +
                "radius=" + radius +
                ", movablePoint=" + movablePoint +
                '}';
    }
}
```

Task 7

```
public class Exercise7 {

    public static void main(String[] args) {
        CircleGeometricObject circleGeometricObject = new
CircleGeometricObject(14);
        System.out.println(circleGeometricObject.getArea());
```

```
        System.out.println(circleGeometricObject.getPerimeter());
        System.out.println(circleGeometricObject);

        Resizable resizable = new ResizableCircle(40);
        resizable.resize(30);
        System.out.println(resizable);
    }
}

interface GeometricObject {
    double getPerimeter();

    double getArea();
}

interface Resizable {
    void resize(int percent);
}

class CircleGeometricObject implements GeometricObject {

    protected float radius;

    public CircleGeometricObject(float radius) {
        this.radius = radius;
    }

    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public String toString() {
        return "CircleGeometricObject{" +
                "radius=" + radius +
                '}';
    }
}

class ResizableCircle extends CircleGeometricObject implements
Resizable {

    public ResizableCircle(float radius) {
```

```
        super(radius);
    }

    @Override
    public void resize(int percent) {
        radius = radius * percent / 100;
    }

    @Override
    public String toString() {
        return "ResizableCircle{" +
                "radius=" + radius +
                '}';
    }
}
```

Exceptions - answers

Task 1

```
public class Exercise1 {
    public static void main(String[] args) throws
CannotDivideBy0Exception {
        MathUtils.divide(10, 0);
    }
}

class CannotDivideBy0Exception extends Exception {
    public CannotDivideBy0Exception() {
        super("Can't divide by 0!");
    }
}

class MathUtils {

    public static float divide(int a, int b) throws
CannotDivideBy0Exception {
        if (b == 0) {
            throw new CannotDivideBy0Exception();
        }
        return a / b;
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) throws
NoBookFoundException {
        BookRepository bookRepository = new BookRepository();
        bookRepository.add(new Book("Harry Potter Part 1", "J.K.
Rowling", "3323-434ds"));
        bookRepository.add(new Book("Harry Potter Part 2", "J.K.
Rowling", "54dsd-dsds"));
        List<Book> book = bookRepository.findByName("??");
```

```
        Book book1 = bookRepository.findByIsbn("??");
        bookRepository.delete("43");
    }
}

class NoBookFoundException extends Exception {
    public NoBookFoundException(String message) {
        super(message);
    }
}

class Book {
    private String title;
    private String author;
    private String isbn;

    public Book(String title, String author, String isbn) {
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getId() {
        return isbn;
    }

    public void setId(String id) {
        this.isbn = id;
    }

    @Override
    public String toString() {
        return "Book{" +
```

```
                "title='" + title + '\'' +
                ", author='" + author + '\'' +
                ", isbn=" + isbn +
                '}';
    }
}

class BookRepository {

    private List<Book> books = new ArrayList<>();

    public void add(Book book) {
        this.books.add(book);
    }

    public void delete(String isbn) throws NoBookFoundException {
        for (Book book : books) {
            if (book.getId().equals(isbn)) {
                books.remove(book);
                return;
            }
        }
        throw new NoBookFoundException("Can't delete book with
isbn: " + isbn);
    }

    public Book findByIsbn(String isbn) throws
NoBookFoundException {
        for (Book book : books) {
            if (book.getId().equals(isbn)) {
                return book;
            }
        }
        throw new NoBookFoundException("Can't find book with isbn:
" + isbn);
    }

    public List<Book> findByName(String name) throws
NoBookFoundException {
        List<Book> booksByName = new ArrayList<>();
        for (Book book : books) {
            if (book.getTitle().equals(name)) {
                booksByName.add(book);
            }
        }
        if (booksByName.isEmpty()) {
            throw new NoBookFoundException("Can't find book with
name: " + name);
        }
```

```
        return booksByName;
    }
}
```

Classes and interfaces - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        UserValidator userValidator = new UserValidator();
        String[] results = userValidator.validateEmails("pb@",
"@yahoo.com");
        System.out.println(results[0]);
        System.out.println(results[1]);
    }
}

class UserValidator {
    public String[] validateEmails(String email, String
alternativeEmail) {
        class Email {
            private String email;

            public Email(String email) {
                if (email == null || email.isEmpty() ||
!validate(email)) {
                    this.email = "unknown";
                } else {
                    this.email = email;
                }
            }
        }
        Email email1 = new Email(email);
        Email email2 = new Email(alternativeEmail);
        return new String[]{email1.email, email2.email};
    }

    public static final Pattern VALID_EMAIL_ADDRESS_REGEX =
            Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]
{2,6}$", Pattern.CASE_INSENSITIVE);

    public static boolean validate(String emailStr) {
        Matcher matcher =
VALID_EMAIL_ADDRESS_REGEX.matcher(emailStr);
```

```
        return matcher.find();
    }
}
```

Task 2

```
public class Exercise2 {
    public static void main(String[] args) {
        Movie movie = new Movie.MovieCreator()
                .setTitle("Star Wars")
                .setDirector("J.J Abrams")
                .setGenre("Action")
                .setYearOfRelease(2015)
                .setPublisher("Disney")
                .createMovie();
        System.out.println(movie);
    }
}

class Movie {
    private String title;
    private String director;
    private int yearOfRelease;
    private String genre;
    private String publisher;

    public Movie(String title, String director, int yearOfRelease,
String genre, String publisher) {
        this.title = title;
        this.director = director;
        this.yearOfRelease = yearOfRelease;
        this.genre = genre;
        this.publisher = publisher;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDirector() {
        return director;
    }
```

```
    public void setDirector(String director) {
        this.director = director;
    }

    public int getYearOfRelease() {
        return yearOfRelease;
    }

    public void setYearOfRelease(int yearOfRelease) {
        this.yearOfRelease = yearOfRelease;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public String getPublisher() {
        return publisher;
    }

    public void setPublisher(String publisher) {
        this.publisher = publisher;
    }

    @Override
    public String toString() {
        return "Movie{" +
                "title='" + title + '\'' +
                ", director='" + director + '\'' +
                ", yearOfRelease='" + yearOfRelease + '\'' +
                ", genre='" + genre + '\'' +
                ", publisher='" + publisher + '\'' +
                '}';
    }

    static class MovieCreator {
        private String title;
        private String director;
        private int yearOfRelease;
        private String genre;
        private String publisher;

        public MovieCreator setTitle(String title) {
            this.title = title;
```

```
            return this;
        }

        public MovieCreator setDirector(String director) {
            this.director = director;
            return this;
        }

        public MovieCreator setYearOfRelease(int yearOfRelease) {
            this.yearOfRelease = yearOfRelease;
            return this;
        }

        public MovieCreator setGenre(String genre) {
            this.genre = genre;
            return this;
        }

        public MovieCreator setPublisher(String publisher) {
            this.publisher = publisher;
            return this;
        }

        public Movie createMovie() {
            Movie movie = new Movie(title, director,
yearOfRelease, genre, publisher);
            return movie;
        }
    }
}
```

Task 3

```
public class Exercise3 {

    public static void main(String[] args) {
        Car car = new Car("VW", "sport");
        System.out.println(car);
    }
}

class Car {
    private String name;
    private String type;
    private Engine engine;
```

```
    public Car(String name, String type) {
        this.name = name;
        this.type = type;
        engine = new Car.Engine();
        engine.setEngineType(type);
    }

    public String getName() {
        return name;
    }

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

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public Engine getEngine() {
        return engine;
    }

    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    @Override
    public String toString() {
        return "Car{" +
                "name='" + name + '\'' +
                ", type='" + type + '\'' +
                ", engine=" + engine +
                '}';
    }

    class Engine {
        private String engineType;

        public void setEngineType(String carType) {
            switch (carType) {
                case "economy":
                    engineType = "diesel";
                    break;
                case "luxury":
```

```
                    engineType = "electric";
                    break;
                default:
                    engineType = "petrol";
            }
        }

        @Override
        public String toString() {
            return "Engine{" +
                    "engineType='" + engineType + '\'' +
                    '}';
        }
    }
}
```

Task 4

```
public class Exercise4 {

    public static void main(String[] args) {
        User user = new User();
        user.setName("John", new Validator<String>() {
            @Override
            public boolean validate(String input) {
                return !input.isEmpty() &&
Character.isUpperCase(input.charAt(0));
            }
        });
        user.setLastName("Smith", new Validator<String>() {
            @Override
            public boolean validate(String input) {
                return input != null && !input.isEmpty() &&
Character.isUpperCase(input.charAt(0));
            }
        });
        user.setAge(20, new Validator<Integer>() {
            @Override
            public boolean validate(Integer input) {
                return input >= 0 && input <= 150;
            }
        });
        user.setLogin("test", new Validator<String>() {
            @Override
            public boolean validate(String input) {
                return input.length() == 10;
```

```
            }
        });
        user.setPassword("test", new Validator<String>() {
            @Override
            public boolean validate(String input) {
                return input.contains("!");
            }
        });
        System.out.println(user);
    }
}

interface Validator<T> {
    boolean validate(T input);
}

class User {
    private String name;
    private String lastName;
    private int age;
    private String login;
    private String password;

    public String getName() {
        return name;
    }

    public void setName(String name, Validator<String> validator)
{
        if (validator.validate(name)) {
            this.name = name;
        }
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName, Validator<String>
validator) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age, Validator<Integer> validator) {
        if (validator.validate(age)) {
```

```
            this.age = age;
        }
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login, Validator<String>
validator) {
        if (validator.validate(login)) {
            this.login = login;
        }
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password, Validator<String>
validator) {
        if (validator.validate(password)) {
            this.password = password;
        }
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", lastName='" + lastName + '\'' +
                ", age=" + age +
                ", login='" + login + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

Enumerated types - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        System.out.println("Saturday is holiday: " +
Weekday.SATURDAY.isHoliday());
        System.out.println("Firday is weekday: " +
Weekday.FRIDAY.isWeekDay());
        Weekday.TUESDAY.whichIsGreater(Weekday.FRIDAY);
    }
}

enum Weekday {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY,
SUNDAY;

    boolean isWeekDay() {
        return this != SATURDAY && this != SUNDAY;
    }

    boolean isHoliday() {
        return this == SATURDAY || this == SUNDAY;
    }

    void whichIsGreater(Weekday weekday) {
        if (this.ordinal() < weekday.ordinal()) {
            System.out.println("Before " + weekday);
        } else {
            System.out.println("After " + weekday);
        }
    }
}
```

Task 2

```
public class Exercise2 {
```

```
    public static void main(String[] args) {
        PackageSize packageSize = PackageSize.getPackageSize(41,
60);
        System.out.println(packageSize);
    }
}

enum PackageSize {
    SMALL(40, 90),
    MEDIUM(90, 140),
    LARGE(140, 250),
    UNKNOWN(0, 0);

    private int minSize;
    private int maxSize;

    PackageSize(int minSize, int maxSize) {
        this.minSize = minSize;
        this.maxSize = maxSize;
    }

    public static PackageSize getPackageSize(int minSize, int
maxSize) {
        for (PackageSize packageSize : values()) {
            if (minSize >= packageSize.minSize && maxSize <
packageSize.maxSize) {
                return packageSize;
            }
        }
        return UNKNOWN;
    }
}
```

Task 3

```
public class Exercise3 {

    public static void main(String[] args) {
        float convertedTemp =
TemperatureConverter.convertTemperature('C', 'K', 34f);
        System.out.println(convertedTemp);
    }
}

interface Converter {
    float convert(float tempIn);
```

```
}

enum TemperatureConverter {
    C_F('C', 'F', new Converter() {
        @Override
        public float convert(float tempIn) {
            return (tempIn * 9 / 5) + 32;
        }
    }),
    C_K('C', 'K', new Converter() {
        @Override
        public float convert(float tempIn) {
            return tempIn + 273.15f;
        }
    }),
    K_C('K', 'C', new Converter() {
        @Override
        public float convert(float tempIn) {
            return tempIn - 273.15f;
        }
    }),
    F_C('F', 'C', new Converter() {
        @Override
        public float convert(float tempIn) {
            return (tempIn - 32) * 5 / 9;
        }
    }),
    F_K('F', 'K', new Converter() {
        @Override
        public float convert(float tempIn) {
            return (tempIn - 32f) * 5 / 9 + 273.15f;
        }
    }),
    K_F('K', 'F', new Converter() {
        @Override
        public float convert(float tempIn) {
            return (tempIn + 273.15f) * 9 / 5 + 32;
        }
    });


    private char input;
    private char output;
    private Converter converter;

    TemperatureConverter(char input, char output, Converter
converter) {
        this.input = input;
        this.output = output;
```

```
        this.converter = converter;
    }

    public static float convertTemperature(char input, char
output, float temp) {
        for (TemperatureConverter temperatureConverter : values())
{
            if (temperatureConverter.input == input &&
temperatureConverter.output == output) {
                return
temperatureConverter.converter.convert(temp);
            }
        }
        return Integer.MIN_VALUE;
    }
}
```

Collections - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        SDAArrayList<Integer> arrayList = new SDAArrayList<>();
        arrayList.add(1);
        arrayList.add(4);
        arrayList.add(5);
        arrayList.add(6);
        arrayList.add(9);
        arrayList.remove(0);
        System.out.println(arrayList.get(0));
        arrayList.display();
    }
}

class SDAArrayList<E> {

    private static final int INITIAL_CAPACITY = 5;
    private Object[] elementArray;
    private int size = 0;


    public SDAArrayList() {
        elementArray = new Object[INITIAL_CAPACITY];
    }

    public E get(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("Index out of
bound exception. Please provide valid index");
        }
        return (E) elementArray[index];
    }


    public void add(E e) {
        if (size == elementArray.length) {
            increaseArraySize();
```

```
        }
        elementArray[size++] = e;
    }

    public E remove(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException("Index out of
bound exception. Please provide valid index");
        }

        Object removedElement = elementArray[index];
        for (int i = index; i < size - 1; i++) {
            elementArray[i] = elementArray[i + 1];
        }
        size--;
        decreaseArraySize();
        return (E) removedElement;
    }


    public void display() {
        for (Object element : elementArray) {
            System.out.println(element);
        }
    }

    private void decreaseArraySize() {
        elementArray = Arrays.copyOf(elementArray,
elementArray.length - 1);
    }

    private void increaseArraySize() {
        int newIncreasedCapacity = elementArray.length * 2;
        elementArray = Arrays.copyOf(elementArray,
newIncreasedCapacity);
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) {
        Author author1 = new Author("John", "Smith", 'M');
        Author author2 = new Author("Jessica", "Albana", 'F');
        Author author3 = new Author("Roger", "Moore", 'M');
```

```
        Author author4 = new Author("Catherin", "Nadie", 'F');

        Book book1 = new Book("Book 1", 34, 2000,
Arrays.asList(author1), Genre.FANTASY);
        Book book2 = new Book("Book 1", 56, 1999,
Arrays.asList(author2, author3, author4), Genre.ACTION);

        BookService bookService = new BookService();
        bookService.add(book1);
        bookService.add(book2);

        System.out.println(bookService.findByAuthor(author2));
        System.out.println(bookService.getAll());
        System.out.println(bookService.findMostExpensiveBook());
        System.out.println(bookService.sortByTitleAsc());
        System.out.println(bookService.sortByTitleDsc());
    }
}

class Author {
    private String name;
    private String lastName;
    private char gender;

    public Author(String name, String lastName, char gender) {
        this.name = name;
        this.lastName = lastName;
        this.gender = gender;
    }

    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public char getGender() {
        return gender;
    }
```

```
    public void setGender(char gender) {
        this.gender = gender;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Author author = (Author) o;
        return gender == author.gender &&
                Objects.equals(name, author.name) &&
                Objects.equals(lastName, author.lastName);
    }

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

    @Override
    public String toString() {
        return "Author{" +
                "name='" + name + '\'' +
                ", lastName='" + lastName + '\'' +
                ", gender=" + gender +
                '}';
    }
}

enum Genre {
    ACTION, FANTASY, CRIME
}

class Book implements Comparable<Book> {
    private String title;
    private float price;
    private int yearOfRelease;
    private List<Author> authors;
    private Genre genre;

    public Book(String title, float price, int yearOfRelease,
List<Author> authors, Genre genre) {
        this.title = title;
        this.price = price;
        this.yearOfRelease = yearOfRelease;
        this.authors = authors;
        this.genre = genre;
    }
```

```
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public float getPrice() {
        return price;
    }

    public void setPrice(float price) {
        this.price = price;
    }

    public int getYearOfRelease() {
        return yearOfRelease;
    }

    public void setYearOfRelease(int yearOfRelease) {
        this.yearOfRelease = yearOfRelease;
    }

    public List<Author> getAuthors() {
        return authors;
    }

    public void setAuthors(List<Author> authors) {
        this.authors = authors;
    }

    public Genre getGenre() {
        return genre;
    }

    public void setGenre(Genre genre) {
        this.genre = genre;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Book book = (Book) o;
        return Float.compare(book.price, price) == 0 &&
                yearOfRelease == book.yearOfRelease &&
                Objects.equals(title, book.title) &&
```

```
                Objects.equals(authors, book.authors) &&
                genre == book.genre;
    }

    @Override
    public int hashCode() {
        return Objects.hash(title, price, yearOfRelease, authors,
genre);
    }

    @Override
    public int compareTo(Book o) {
        return o.getTitle().compareTo(title);
    }

    @Override
    public String toString() {
        return "Book{" +
                "title='" + title + '\'' +
                ", price=" + price +
                ", yearOfRelease=" + yearOfRelease +
                ", authors=" + authors +
                ", genre=" + genre +
                '}';
    }
}

class BookService {
    private List<Book> books = new ArrayList<>();

    public void add(Book book) {
        books.add(book);
    }

    public void remove(Book book) {
        books.remove(book);
    }

    public List<Book> getAll() {
        return books;
    }

    public List<Book> findByGenre(Genre genre) {
        List<Book> results = new ArrayList<>();
        for (Book book : books) {
            if (book.getGenre() == genre) {
                results.add(book);
            }
        }
```

```
        return results;
    }

    public List<Book> findByYear(int yearOfRelease) {
        List<Book> results = new ArrayList<>();
        for (Book book : books) {
            if (book.getYearOfRelease() < yearOfRelease) {
                results.add(book);
            }
        }
        return results;
    }

    public Book findMostExpensiveBook() {
        Book result = null;
        for (Book book : books) {
            if (result == null || result.getPrice() <
book.getPrice()) {
                result = book;
            }
        }
        return result;
    }

    public Book findCheapestBook() {
        Book result = null;
        for (Book book : books) {
            if (result == null || result.getPrice() >
book.getPrice()) {
                result = book;
            }
        }
        return result;
    }

    public List<Book> findByNumberOfAuthors(int numberOfAuthors) {
        List<Book> results = new ArrayList<>();
        for (Book book : books) {
            if (book.getAuthors().size() == numberOfAuthors) {
                results.add(book);
            }
        }
        return results;
    }

    public List<Book> sortByTitleAsc() {
        Collections.sort(books);
        return books;
    }
```

```
    public List<Book> sortByTitleDsc() {
        Collections.reverse(books);
        return books;
    }

    public boolean isBookInRepo(Book book) {
        return books.contains(book);
    }

    public List<Book> findByAuthor(Author author) {
        List<Book> results = new ArrayList<>();
        for (Book book : books) {
            if (book.getAuthors().contains(author)) {
                results.add(book);
            }
        }
        return results;
    }
}
```

Task 3

```
public class Exercise3 {

    public static void main(String[] args) {
        Random random = new Random();
        List<Integer> values = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            values.add(random.nextInt(50));
        }
        Set<Integer> uniqueValues = new HashSet<>();
        Set<Integer> duplicatedValues = new HashSet<>();
        for (Integer value : values) {
            if (!uniqueValues.add(value)) {
                duplicatedValues.add(value);
            }
        }
        System.out.println("Unique: " + uniqueValues);
        System.out.println("Duplicated: " + duplicatedValues);
    }
}
```

Task 4

```
public class Exercise4 {
    public static void main(String[] args) {
        Author author1 = new Author("John", "Smith", 'M');
        Author author2 = new Author("Jessica", "Albana", 'F');
        Author author3 = new Author("Roger", "Moore", 'M');
        Author author4 = new Author("Catherin", "Nadie", 'F');

        Book book1 = new Book("Book 1", 34, 2000,
Arrays.asList(author1), Genre.FANTASY);
        Book book2 = new Book("Book 1", 56, 1999,
Arrays.asList(author2, author3, author4), Genre.ACTION);

        BookServiceExt bookService = new BookServiceExt();
        bookService.add(book1);
        bookService.add(book2);

        System.out.println(bookService.mapBooks());
    }
}

class BookServiceExt {
    private List<Book> books = new ArrayList<>();

    public void add(Book book) {
        books.add(book);
    }

    public void remove(Book book) {
        books.remove(book);
    }

    public List<Book> getAll() {
        return books;
    }

    public Map<Genre, String> mapBooks() {
        Map<Genre, String> booksMap = new HashMap<>();
        for (Book book : books) {
            booksMap.put(book.getGenre(), book.getTitle());
        }
        return booksMap;
    }

}
```

Task 5

```
public class Exercise5 {

    public static void main(String[] args) {
        Author author1 = new Author("John", "Smith", 'M');
        Author author2 = new Author("Jessica", "Albana", 'F');
        Author author3 = new Author("Roger", "Moore", 'M');
        Author author4 = new Author("Catherin", "Nadie", 'F');

        Book book1 = new Book("Book 1", 34, 2000,
Arrays.asList(author1), Genre.FANTASY);
        Book book2 = new Book("Book 1", 56, 1999,
Arrays.asList(author2, author3, author4), Genre.ACTION);

        BookServiceExt2 bookService = new BookServiceExt2();
        bookService.add(book1);
        bookService.add(book2);

        Stack<Book> bookStack = bookService.createBookStack();
        while (!bookStack.isEmpty()) {
            System.out.println(bookStack.pop());
        }
    }

}

class BookServiceExt2 {
    private List<Book> books = new ArrayList<>();

    public void add(Book book) {
        books.add(book);
    }

    public void remove(Book book) {
        books.remove(book);
    }

    public List<Book> getAll() {
        return books;
    }

    public Stack<Book> createBookStack() {
        Collections.sort(books, Comparator.comparingDouble(new
ToDoubleFunction<Book>() {
            @Override
            public double applyAsDouble(Book value) {
```

```
                return value.getPrice();
            }
        }));
        Stack<Book> bookStack = new Stack<>();
        for (Book book : books) {
            bookStack.push(book);
        }
        return bookStack;
    }

}
```

Functional programming - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        Video video = new Video("GOT1", "got1.com",
VideoType.CLIP);
        Video video1 = new Video("GOT2", "got2.com",
VideoType.EPISODE);
        Video video2 = new Video("GOT3", "got3.com",
VideoType.PREVIEW);
        Video video3 = new Video("GOT4", "got4.com",
VideoType.PREVIEW);
        Video video4 = new Video("GOT5", "got5.com",
VideoType.CLIP);
        Video video5 = new Video("GOT6", "got6.com",
VideoType.EPISODE);

        Episode episode = new Episode("got1", 1,
                Arrays.asList(video, video1));
        Episode episode1 = new Episode("got2", 2,
                Arrays.asList(video2, video3));
        Episode episode2 = new Episode("got3", 1,
                Arrays.asList(video4, video5));
        Season season = new Season("GOTS1", 1,
                Arrays.asList(episode, episode1));
        Season season1 = new Season("GOTS1", 2,
                Arrays.asList(episode2));

        List<Season> seasons = Arrays.asList(season, season1);

        //list of episodes
        List<Episode> episodes = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .collect(Collectors.toList());

        //list of videos
        List<Video> videos = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .flatMap(e -> e.videos.stream())
```

```
                .collect(Collectors.toList());

        //list of seasons names
        List<String> seasonNames = seasons.stream()
                .map(s -> s.seasonName)
                .collect(Collectors.toList());

        //list of seasons numbers
        List<Integer> seasonNumbers = seasons.stream()
                .map(s -> s.seasonNumber)
                .collect(Collectors.toList());

        //list of episodes names
        List<String> episodeNames = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .map(e -> e.episodeName)
                .collect(Collectors.toList());

        //list of episodes numbers
        List<Integer> episodeNumbers = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .map(e -> e.episodeNumber)
                .collect(Collectors.toList());

        //list of videos names
        List<String> videoNames = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .flatMap(e -> e.videos.stream())
                .map(v -> v.title)
                .collect(Collectors.toList());

        //list of videos urls
        List<String> videoUrls = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .flatMap(e -> e.videos.stream())
                .map(v -> v.url)
                .collect(Collectors.toList());

        //list of even episodes
        List<Episode> evenEpisodes = seasons.stream()
                .flatMap(s -> season.episodes.stream())
                .filter(e -> e.episodeNumber % 2 == 0)
                .collect(Collectors.toList());

        //list of even seasons
        List<Season> evenSeasons = seasons.stream()
                .filter(s -> s.seasonNumber % 2 == 0)
                .collect(Collectors.toList());
```

```
        //list of even episodes and seasons
        List<Episode> evenEpisodesFromEvenSeasons =
seasons.stream()
                .filter(s -> s.seasonNumber % 2 == 0)
                .flatMap(s -> season.episodes.stream())
                .filter(e -> e.episodeNumber % 2 == 0)
                .collect(Collectors.toList());

        //list of clips videos from even episodes and odd seasons
        List<Video> clipVideoFromEvenEpisodesFromOddSeasons =
seasons.stream()
                .filter(s -> s.seasonNumber % 2 == 0)
                .flatMap(s -> season.episodes.stream())
                .filter(e -> e.episodeNumber % 2 != 0)
                .flatMap(e -> e.videos.stream())
                .filter(v -> v.videoType == VideoType.CLIP)
                .collect(Collectors.toList());

        //list of preview videos from odd episodes and even
seasons
        List<Video> previewVideoFromOddEpisodesFromEvenSeasons =
seasons.stream()
                .filter(s -> s.seasonNumber % 2 != 0)
                .flatMap(s -> season.episodes.stream())
                .filter(e -> e.episodeNumber % 2 == 0)
                .flatMap(e -> e.videos.stream())
                .filter(v -> v.videoType == VideoType.PREVIEW)
                .collect(Collectors.toList());
    }
}

enum VideoType {
    CLIP, PREVIEW, EPISODE
}

class Video {
    public String title;
    public String url;
    public VideoType videoType;

    public Video(String title, String url, VideoType videoType) {
        this.title = title;
        this.url = url;
        this.videoType = videoType;
    }

    @Override
    public String toString() {
        return "Video{" +
```

```
                "title='" + title + '\'' +
                ", url='" + url + '\'' +
                ", videoType=" + videoType +
                '}';
    }
}

class Episode {
    public String episodeName;
    public int episodeNumber;
    List<Video> videos;

    public Episode(String episodeName, int episodeNumber,
List<Video> videos) {
        this.episodeName = episodeName;
        this.episodeNumber = episodeNumber;
        this.videos = videos;
    }

    @Override
    public String toString() {
        return "Episode{" +
                "episodeName='" + episodeName + '\'' +
                ", episodeNumber=" + episodeNumber +
                ", videos=" + videos +
                '}';
    }
}

class Season {
    public String seasonName;
    public int seasonNumber;
    List<Episode> episodes;

    public Season(String seasonName, int seasonNumber,
List<Episode> episodes) {
        this.seasonName = seasonName;
        this.seasonNumber = seasonNumber;
        this.episodes = episodes;
    }

    @Override
    public String toString() {
        return "Season{" +
                "seasonName='" + seasonName + '\'' +
                ", seasonNumber=" + seasonNumber +
                ", episodes=" + episodes +
                '}';
```

```
        }
    }
```

Generic types - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        Pair<Integer, String> pair = new Pair<>(23,
"JavaAdvanced");
        System.out.println(pair);
    }
}

class Pair<K, V> {
    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }

    @Override
    public String toString() {
        return "Pair{" +
                "key=" + key +
```

```
                ", value=" + value +
                '}';
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) {
        Integer[] tab = {10, 21, 33, 40, 50, 60};
        int counter = Utils.countIf(tab, new Validator<Integer>()
{
            @Override
            public boolean validate(Integer value) {
                return value % 3 == 0;
            }
        });
        System.out.println(counter);
    }
}

interface Validator<T> {
    boolean validate(T value);
}

class Utils {
    public static <T> int countIf(T[] tab, Validator<T> validator)
{
        int counter = 0;
        for (T element : tab) {
            if (validator.validate(element)) {
                counter++;
            }
        }
        return counter;
    }
}
```

Task 3

```
public class Exercise3 {
```

```
    public static void main(String[] args) {
        Integer[] tab = {10, 21, 33, 40, 50, 60};
        ArrayUtils.swap(tab, 2, 5);
        System.out.println(Arrays.toString(tab));
    }
}

class ArrayUtils {

    public static <T> void swap(T[] array, int index1, int index2)
{
        T tmp = array[index1];
        array[index1] = array[index2];
        array[index2] = tmp;
    }
}
```

Task 4

```
public class Exercise4 {

    public static void main(String[] args) {
        Library<Book> bookLibrary = new Library<>(new Book[]{new
Book("Harry Potter", "Fantasy")});

System.out.println(Arrays.toString(bookLibrary.getElements()));

        Library<Movie> movieLibrary = new Library<>(new Movie[]
{new Movie("Star Wars", "J.J Ambrams")});

System.out.println(Arrays.toString(movieLibrary.getElements()));

        Library<Newspaper> newspaperLibrary = new Library<>(new
Newspaper[]{new Newspaper("NYC", "US")});

System.out.println(Arrays.toString(newspaperLibrary.getElements()));

    }
}

abstract class MediaContent {
    protected String title;

    public MediaContent(String title) {
        this.title = title;
    }
```

```
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}

class Book extends MediaContent {
    private String author;

    public Book(String title, String author) {
        super(title);
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    @Override
    public String toString() {
        return "Book{" +
                "title='" + title + '\'' +
                ", author='" + author + '\'' +
                '}';
    }
}

class Newspaper extends MediaContent {

    private String editor;

    public Newspaper(String title, String editor) {
        super(title);
        this.editor = editor;
    }

    public String getEditor() {
        return editor;
    }

    public void setEditor(String editor) {
```

```
        this.editor = editor;
    }

    @Override
    public String toString() {
        return "Newspaper{" +
                "title='" + title + '\'' +
                ", editor='" + editor + '\'' +
                '}';
    }
}

class Movie extends MediaContent {

    private String director;

    public Movie(String director, String title) {
        super(title);
        this.director = director;
    }

    public String getDirector() {
        return director;
    }

    public void setDirector(String director) {
        this.director = director;
    }

    @Override
    public String toString() {
        return "Movie{" +
                "title='" + title + '\'' +
                ", director='" + director + '\'' +
                '}';
    }
}

class Library<T extends MediaContent> {

    private T[] elements;

    public Library(T[] elements) {
        this.elements = elements;
    }

    public T[] getElements() {
        return elements;
    }
```

```
    public void setElements(T[] elements) {
        this.elements = elements;
    }
}
```

Task 5

```
public class Exercise5 {

    public static void main(String[] args) {
        Animal[] animal = {new Cat("Persian", 10), new Dog("German
Shepherd", "beef")};
        AnimalHouse<Animal> animals = new AnimalHouse<>(animal);
        System.out.println(Arrays.toString(animals.getAnimals()));
    }
}

abstract class Animal {
    protected String name;

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

    public String getName() {
        return name;
    }

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

class Dog extends Animal {

    private String favoriteFood;

    public Dog(String name, String favoriteFood) {
        super(name);
        this.favoriteFood = favoriteFood;
    }

    public String getFavoriteFood() {
        return favoriteFood;
```

```
    }

    public void setFavoriteFood(String favoriteFood) {
        this.favoriteFood = favoriteFood;
    }

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

class Cat extends Animal {

    private int numberOfLife;

    public Cat(String name, int numberOfLife) {
        super(name);
        this.numberOfLife = numberOfLife;
    }

    public int getNumberOfLife() {
        return numberOfLife;
    }

    public void setNumberOfLife(int numberOfLife) {
        this.numberOfLife = numberOfLife;
    }

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

class AnimalHouse<T extends Animal> {

    T[] animals;

    public AnimalHouse(T[] animals) {
        this.animals = animals;
    }
```

```
    public T[] getAnimals() {
        return animals;
    }

    public void setAnimals(T[] animals) {
        this.animals = animals;
    }
}
```

Java IO - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        //replace with sample directory path on Your OS
        File file = new File("/Users/sdauser/Documents/sda");
        String[] fileList = file.list();
        for (String name : fileList) {
            System.out.println(name);
        }
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) {
        BufferedReader bufferedReader;
        String strLine;
        try {
            bufferedReader = new BufferedReader(new
FileReader("/Users/sdauser/Documents/sda/code/test.txt"));
            while ((strLine = bufferedReader.readLine()) != null)
{
                System.out.println(strLine);
            }
            bufferedReader.close();
        } catch (FileNotFoundException e) {
            System.err.println("File not found");
        } catch (IOException e) {
            System.err.println("Unable to read the file.");
        }
    }
}
```

Task 3

```
public class Exercise3 {

    public static void main(String[] args) {
        StringBuilder stringBuilder = new StringBuilder();
        String strLine = "";
        try {
            String filename =
"/Users/sdauser/Documents/sda/code/test.txt";
            FileWriter fw = new FileWriter(filename, true);
            fw.write("Java I/O Exercises\n");
            fw.close();
            BufferedReader br = new BufferedReader(new
FileReader(filename));
            while (strLine != null) {
                stringBuilder.append(strLine);
                stringBuilder.append(System.lineSeparator());
                strLine = br.readLine();
                System.out.println(strLine);
            }
            br.close();
        } catch (IOException ioe) {
            System.err.println("IOException: " +
ioe.getMessage());
        }
    }
}
```

Task 4

```
public class Exercise4 {

    public static void main(String[] args) throws
FileNotFoundException {
        String longestWord = new Exercise4().findLongestWords();
        System.out.println(longestWord);
    }

    public String findLongestWords() throws FileNotFoundException
{
        String longestWord = "";
        String current;
        Scanner scanner = new Scanner(new
```

```
File("/Users/sdauser/Documents/sda/code/test.txt"));
        while (scanner.hasNext()) {
            current = scanner.next();
            if (current.length() > longestWord.length()) {
                longestWord = current;
            }

        }
        return longestWord;
    }
}
```

Task 5

```
public class Exercise5 {
    public static void main(String[] args) throws IOException {
        UserParser userParser = new UserParser();
        Path path =
Paths.get("/Users/sdauser/Documents/sda/code/test.txt");
        List<User> users = new ArrayList<>();
        List<String> lines = Files.readAllLines(path);
        for (String line : lines) {
            User user = userParser.fromCSV(line);
            users.add(user);
        }
        System.out.println("Results " + users);
    }
}

class UserParser {
    public User fromCSV(String csvLine) {
        String[] data = csvLine.split(",");
        return new User(data[0], data[1],
Integer.parseInt(data[2]));
    }
}

class User {
    private String name;
    private String lastName;
    private int age;

    public User(String name, String lastName, int age) {
        this.name = name;
        this.lastName = lastName;
        this.age = age;
```

```
    }

    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getAge() {
        return age;
    }

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

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

Task 6

```
public class Exercise6 {

    public static void main(String[] args) throws IOException {
        MovieFileRepository movieFileRepository = new
MovieFileRepository();
        movieFileRepository.add(new Movie("Star Wars Force
Awaken", "Action", "J.J Ambrams", 2015));
        movieFileRepository.add(new Movie("Star Wars Last Jedi",
```

```
"Action", "J.J Ambrams", 2017));
        System.out.println("Results :" +
movieFileRepository.getAll());
    }
}

class MovieFileRepository {

    private final MovieParser movieParser = new MovieParser();
    private final static Path PATH =
Paths.get("/Users/sdauser/Documents/sda/code/test.txt");

    public void add(Movie movie) throws IOException {
        Files.writeString(PATH, movieParser.toCSV(movie),
StandardOpenOption.APPEND);
    }

    public List<Movie> getAll() throws IOException {
        List<String> movieLines = Files.readAllLines(PATH);
        List<Movie> movies = new ArrayList<>();
        for (String line : movieLines) {
            Movie movie = movieParser.fromCSV(line);
            movies.add(movie);
        }
        return movies;
    }
}

class MovieParser {
    private static final String SEPARATOR = ",";

    public Movie fromCSV(String line) {
        String[] data = line.split(SEPARATOR);
        return new Movie(data[0], data[1], data[2],
Integer.parseInt(data[3]));
    }

    public String toCSV(Movie movie) {
        return new StringBuilder().append(movie.getTitle())
                .append(SEPARATOR)
                .append(movie.getGenre())
                .append(SEPARATOR)
                .append(movie.getDirector())
                .append(SEPARATOR)
                .append(movie.getYearOfRelease())
                .append("\n")
                .toString();
    }
}
```

```
class Movie {
    private String title;
    private String genre;
    private String director;
    private int yearOfRelease;

    public Movie(String title, String genre, String director, int
yearOfRelease) {
        this.title = title;
        this.genre = genre;
        this.director = director;
        this.yearOfRelease = yearOfRelease;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public String getDirector() {
        return director;
    }

    public void setDirector(String director) {
        this.director = director;
    }

    public int getYearOfRelease() {
        return yearOfRelease;
    }

    public void setYearOfRelease(int yearOfRelease) {
        this.yearOfRelease = yearOfRelease;
    }

    @Override
    public String toString() {
```

```
        return "Movie{" +
                "title='" + title + '\'' +
                ", genre='" + genre + '\'' +
                ", director='" + director + '\'' +
                ", yearOfRelease=" + yearOfRelease +
                '}';
    }
}
```

Parallel and concurrent programming - answers

Task 1

```
public class Exercise1 {

    public static void main(String[] args) {
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 1000; i < 2000; i++) {
                    if (i % 2 == 0) {

System.out.println(Thread.currentThread().getName() + " " + i);
                    }
                }
            }
        });

        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 14300; i < 17800; i++) {
                    if (i % 2 == 0) {

System.out.println(Thread.currentThread().getName() + " " + i);
                    }
                }
            }
        });

        thread1.start();
        thread2.start();

    }

}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) {
        Bridge bridge = new Bridge();
        Car car1 = new Car("VW", "Combi");
        Car car2 = new Car("SEAT", "Suv");
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                bridge.driveThrough(car1);
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                bridge.driveThrough(car2);
            }
        });
        thread1.start();
        thread2.start();
    }
}

class Car {
    private String name;
    private String type;

    public Car(String name, String type) {
        this.name = name;
        this.type = type;
    }

    public String getName() {
        return name;
    }

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

    public String getType() {
        return type;
    }

    public void setType(String type) {
```

```
        this.type = type;
    }

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

class Bridge {

    public synchronized void driveThrough(Car car) {
        System.out.println("Driving through: " + car);
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Finished!: " + car);
    }
}
```

Task 3

```
public class Exercise3 {
    public static void main(String[] args) throws
InterruptedException, ExecutionException {
        Random random = new Random();
        int[] array1 = new int[10000];
        int[] array2 = new int[10000];
        for (int i = 0; i < 10000; i++) {
            array1[i] = random.nextInt(20000);
            array2[i] = array1[i];
        }
        ExecutorService executorService =
Executors.newFixedThreadPool(2);
        String result = executorService.invokeAny(Arrays.asList(
                new BubbleSortStrategy(array1),
                new InsertionSortStrategy(array2)
        ));
        System.out.println(result);
        executorService.shutdown();
    }
```

```
}

class BubbleSortStrategy implements Callable<String> {

    private int[] array;

    public BubbleSortStrategy(int[] array) {
        this.array = array;
    }

    public void bubbleSort() {
        boolean sorted = false;
        int temp;
        while (!sorted) {
            sorted = true;
            for (int i = 0; i < array.length - 1; i++) {
                if (array[i] > array[i + 1]) {
                    temp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = temp;
                    sorted = false;
                }
            }
        }
    }

    @Override
    public String call() throws Exception {
        bubbleSort();
        return "Bubble sort";
    }
}

class InsertionSortStrategy implements Callable<String> {

    private int[] array;

    public InsertionSortStrategy(int[] array) {
        this.array = array;
    }

    public void insertionSort() {
        for (int i = 1; i < array.length; i++) {
            int current = array[i];
            int j = i - 1;
            while (j >= 0 && current < array[j]) {
                array[j + 1] = array[j];
                j--;
            }
```

```
            array[j + 1] = current;
        }
    }

    @Override
    public String call() throws Exception {
        insertionSort();
        return "Insertion sort";
    }
}
```

Task 4

```
public class Exercise4 {

    public static void main(String[] args) {
        Account account = new Account(10000);

        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    account.pay(20000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                account.transfer(5000);
            }
        });

        Thread thread3 = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(4000);
```

```
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                account.transfer(6000);
            }
        });

        thread1.start();
        thread2.start();
        thread3.start();
    }
}

class Account {

    private float saldo;

    public Account(int saldo) {
        this.saldo = saldo;
    }

    synchronized void transfer(float amount) {
        saldo += amount;
        notify();
        System.out.println(String.format("Transfer %f, saldo: %f",
amount, saldo));
    }

    synchronized void pay(float amount) throws
InterruptedException {
        while (amount > saldo) {
            System.out.println("Not enough money! Waiting ... ");
            wait();
        }
        saldo -= amount;
        System.out.println(String.format("Pay %f, saldo: %f",
amount, saldo));
    }

}
```

Task 5

```
public class Exercise5 {

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

```
        Iterator<Integer> iterator = new Iterator<Integer>(new
Integer[]{1, 4, 50, 434, 78});
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    int value = iterator.next();

System.out.println(Thread.currentThread().getName() + " " +
value);
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    int value = iterator.prev();

System.out.println(Thread.currentThread().getName() + " " +
value);
                    try {
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        thread1.start();
        thread2.start();
    }
}

class Iterator<T> {

    private AtomicInteger atomicInteger = new AtomicInteger(0);
    private T[] data;

    public Iterator(T[] data) {
        this.data = data;
    }

    public T next() {
```

```
        if (atomicInteger.get() < data.length) {
            return data[atomicInteger.getAndIncrement()];
        }
        throw new IllegalArgumentException("Out of range!");
    }

    public T prev() {
        if (atomicInteger.get() > 0 && atomicInteger.get() <
data.length) {
            return data[atomicInteger.getAndDecrement()];
        }
        throw new IllegalArgumentException("Out of range!");
    }
}
```

Reflection API basics - answers

Task 1

```
public class Exercise1 {
    public static void main(String[] args) {
        Student student = new Student();

        System.out.println("Methods: ");
        Method[] methods =
student.getClass().getDeclaredMethods();
        System.out.println(Arrays.toString(methods));

        System.out.println("Fields: ");
        Field[] fields = student.getClass().getDeclaredFields();
        System.out.println(Arrays.toString(fields));

        System.out.println("Constructors: ");
        Constructor[] constructors =
student.getClass().getConstructors();
        System.out.println(Arrays.asList(constructors));
    }
}

class Student {
    private String name;
    private String lastName;
    private int index;
    private String typeOfStudies;

    public Student() {

    }

    public Student(String name, String lastName, int index, String
typeOfStudies) {
        this.name = name;
        this.lastName = lastName;
        this.index = index;
        this.typeOfStudies = typeOfStudies;
    }
```

```
    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public String getTypeOfStudies() {
        return typeOfStudies;
    }

    public void setTypeOfStudies(String typeOfStudies) {
        this.typeOfStudies = typeOfStudies;
    }
}
```

Task 2

```
public class Exercise2 {

    public static void main(String[] args) throws
ClassNotFoundException, NoSuchMethodException,
IllegalAccessException, InvocationTargetException,
InstantiationException, NoSuchFieldException {
        StudentExt studentExt = (StudentExt)
Class.forName("reflection_api.StudentExt")
                .getConstructor(String.class, String.class,
Integer.class, String.class)
                .newInstance("John", "Smith", 10, "It");
```

```
        System.out.println(studentExt);

        Field nameField =
studentExt.getClass().getDeclaredField("name");
        nameField.setAccessible(true);
        nameField.set(studentExt, "Johnson");

        Field lastNameField =
studentExt.getClass().getDeclaredField("lastName");
        lastNameField.setAccessible(true);
        lastNameField.set(studentExt, "Spring");

        String name = (String)
studentExt.getClass().getMethod("getName").invoke(studentExt);
        System.out.println(name);

        String lastName = (String)
studentExt.getClass().getMethod("getLastName").invoke(studentExt);
        System.out.println(lastName);

        String typeOfStudies = (String)
studentExt.getClass().getMethod("getTypeOfStudies").invoke(studentEx

        System.out.println(typeOfStudies);

        int index = (Integer)
studentExt.getClass().getMethod("getIndex").invoke(studentExt);
        System.out.println(index);
    }
}

class StudentExt {
    private String name;
    private String lastName;
    private int index;
    private String typeOfStudies;

    public StudentExt() {

    }

    public StudentExt(String name, String lastName, Integer index,
String typeOfStudies) {
        this.name = name;
        this.lastName = lastName;
        this.index = index;
        this.typeOfStudies = typeOfStudies;
    }
```

```
    public String getName() {
        return name;
    }

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

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public String getTypeOfStudies() {
        return typeOfStudies;
    }

    public void setTypeOfStudies(String typeOfStudies) {
        this.typeOfStudies = typeOfStudies;
    }

    @Override
    public String toString() {
        return "StudentExt{" +
                "name='" + name + '\'' +
                ", lastName='" + lastName + '\'' +
                ", index=" + index +
                ", typeOfStudies='" + typeOfStudies + '\'' +
                '}';
    }
}
```