Министерство образования Тульской области

Государственное профессиональное образовательное учреждение Тульской области «Донской политехнический колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

по междисциплинарному курсу

«МДК.01.03 Разработка мобильных приложений»

по теме: «Разработка приложения с использованием классов и объектов»

для обучающихся по программе подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование, квалификация «Программист»

Автор:

С.М. Гвоздев, преподаватель ГПОУ ТО «ДПК»

2024 г.

Лист согласования:

Автор разработки:

Гвоздев Сергей Михайлович, преподаватель ГПОУ ТО «ДПК»

Рецензенты:

Евтехова О.А., заместитель директора по учебной и научно-методической работе ГПОУ ТО «ДПК»

Панченко Т.А., заместитель директора по организации образовательного процесса ГПОУ ТО «ДПК».

Филатова Е.А., старший методист ГПОУ ТО «ДПК».

Методические рекомендации предназначены для студентов 3 курса, обучающихся по специальности 09.02.07 Информационные системы и программирование, квалификация «Программист». Конкретные примеры данного пособия окажут практическую помощь при выполнении практических заданий по дисциплине «МДК.01.03 Разработка мобильных приложений»: «Разработка приложения с использованием классов и объектов» с использованием онлайн-редактора кода Dartpad.

СОГЛАСОВАНО

на заседании предметной (цикловой) комиссии
дисциплин профессионального цикла отделения
«Информационная безопасность и администрирование»
Протокол № 2

от «03» октября 2024 г.

Председатель ПЦК Гвоздев С.М.

СОДЕРЖАНИЕ

Введение

Методические рекомендации составлены в соответствии с рабочей программой ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» специальности 09.02.07 Информационные системы и программирование, квалификация «Программист».

В ходе освоения профессионального модуля обучающийся должен:

знать:

- основные этапы разработки программного обеспечения;
- основные принципы технологии структурного и объектно-ориентированного программирования;

уметь:

- разрабатывать приложения на языке Dart с использованием классов и объектов.

В процессе обучения по «МДК.01.03 Разработка мобильных приложений» у студентов формируется комплексное понимание и практические навыки, необходимые для создания функциональных и удобных мобильных приложений для различных платформ, таких как iOS и Android. Это достигается через изучение теоретических основ, включая архитектуру мобильных приложений, принципы работы мобильных операционных систем, основные компоненты приложений, такие как пользовательский интерфейс и бизнес-логика, а также различные паттерны проектирования.

На практике студенты осваивают инструменты и технологии разработки, включая языки программирования, интегрированные среды разработки (IDE), системы управления версиями и другие полезные инструменты. Они учатся проектировать интуитивно понятный и привлекательный пользовательский интерфейс, оптимизировать производительность приложений, обеспечивать безопасность данных и взаимодействие с различными сервисами и API.

Кроме того, студенты получают навыки тестирования и отладки мобильных приложений, а также знакомятся с процессом публикации приложений в магазинах приложений и монетизации. В целом, цель дисциплины – подготовить студентов к успешной карьере в области разработки мобильных приложений, обеспечив их глубоким пониманием технологий и практическими навыками, необходимыми для создания качественных и востребованных продуктов.

Задание № 1: Создание простого класса.

Создайте класс *Person* с двумя свойствами: *name (строка)* и *age (целое число)*. Добавьте конструктор, который принимает эти параметры и инициализирует свойства. Создайте объект класса *Person* и выведите его свойства.

Описание алгоритма решения:

1. Определите класс *Person*.
2. Добавьте в класс два свойства: *name и age*.
3. Создайте конструктор, который принимает параметры *name и age* и инициализирует соответствующие свойства.
4. Создайте объект класса *Person*, передав ему имя и возраст.
5. Выведите свойства объекта с помощью функции *print()*.

Рисунок №1 – Результат решения задания №1.

Задание № 2: Использование методов класса.

Добавьте в класс *Person* метод *sayHello*, который выводит приветствие с именем пользователя. Вызовите этот метод для объекта класса *Person*.

Описание алгоритма решения:

1. Добавьте в класс *Person* метод *sayHello*.
2. В методе *sayHello* используйте функцию *print()* для вывода приветствия с именем пользователя.
3. Вызовите метод *sayHello* для объекта класса Person.

Рисунок №2 – Результат решения задания №2.

Задание № 3: Наследование классов

Создайте класс *Student*, который наследуется от класса *Person*. Добавьте в класс *Student* свойство *studentId (строка) и метод study*, который выводит сообщение о том, что студент учится.

Описание алгоритма решения:

1. Определите класс *Student*, который наследуется от класса *Person*.
2. Добавьте в класс *Student* свойство *studentId*.
3. Создайте конструктор для класса *Student*, который принимает параметры *name, age и studentId*, и вызывает конструктор родительского класса.
4. Добавьте в класс *Student* метод *study*, который выводит сообщение о том, что студент учится.
5. Создайте объект класса *Student* и вызовите его методы *sayHello и study*.

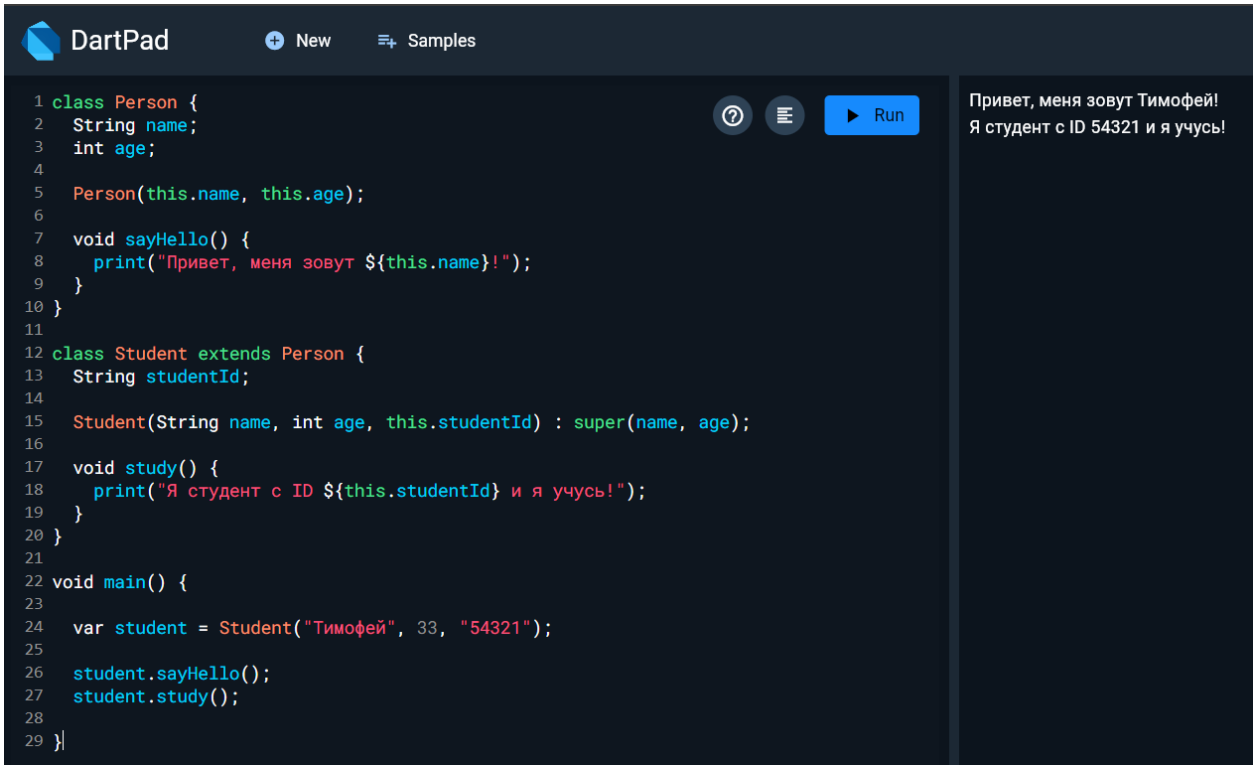


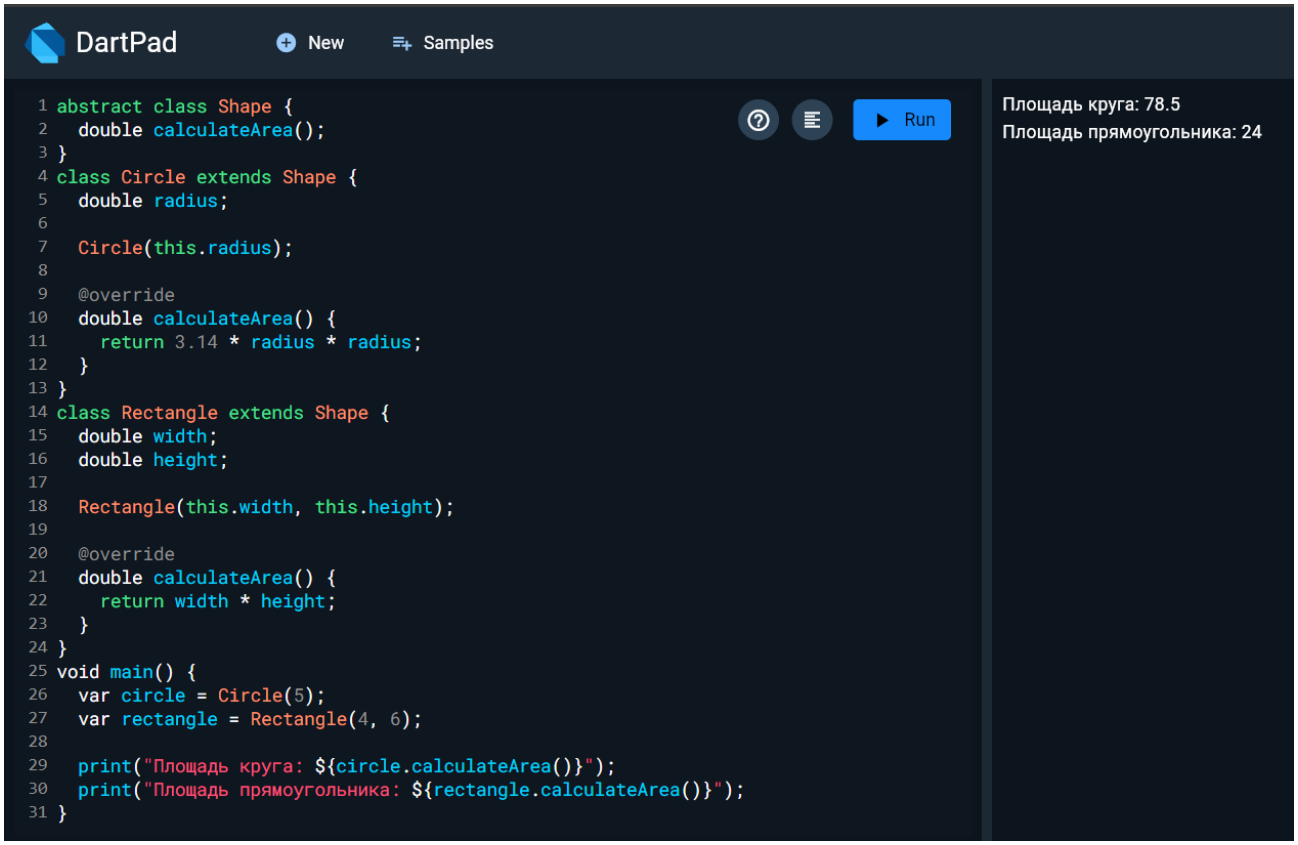
Рисунок №3 – Результат решения задания №3.

Задание № 4: Абстрактные классы и методы

Создайте абстрактный класс *Shape* с абстрактным методом *calculateArea*. Создайте два класса *Circle* и *Rectangle*, которые наследуются от класса *Shape* и реализуют метод *calculateArea*.

Описание алгоритма решения:

1. Определите абстрактный класс *Shape*.
2. Добавьте в класс *Shape* абстрактный метод *calculateArea*.
3. Определите класс *Circle*, который наследуется от класса *Shape*.
4. Добавьте в класс *Circle* свойство *radius*.
5. Реализуйте метод *calculateArea* в классе *Circle*, который возвращает площадь круга.
6. Определите класс *Rectangle*, который наследуется от класса *Shape*.
7. Добавьте в класс *Rectangle* свойства *width и height*.
8. Реализуйте метод *calculateArea* в классе *Rectangle*, который возвращает площадь прямоугольника.
9. Создайте объекты классов *Circle* и *Rectangle* и вызовите их методы *calculateArea*




```dart
abstract class Shape {
  double calculateArea();
}
class Circle extends Shape {
  double radius;

  Circle(this.radius);

  @override
  double calculateArea() {
    return 3.14 * radius * radius;
  }
}
class Rectangle extends Shape {
  double width;
  double height;

  Rectangle(this.width, this.height);

  @override
  double calculateArea() {
    return width * height;
  }
}
void main() {
  var circle = Circle(5);
  var rectangle = Rectangle(4, 6);

  print("Площадь круга: ${circle.calculateArea()}");
  print("Площадь прямоугольника: ${rectangle.calculateArea()}");
}
```

Рисунок №4 – Результат решения задания №4.

Задание № 5: Интерфейсы

Создайте интерфейс *Drawable* с методом *draw*. Создайте класс *Circle*, который реализует интерфейс *Drawable* и выводит сообщение о том, что круг нарисован.

Описание алгоритма решения:

1. Определите интерфейс *Drawable* с методом *draw*.
2. Определите класс *Circle*, который реализует интерфейс *Drawable*.
3. Реализуйте метод *draw* в классе *Circle*, который выводит сообщение о том, что круг нарисован.
4. Создайте объект класса *Circle* и вызовите его метод *draw*

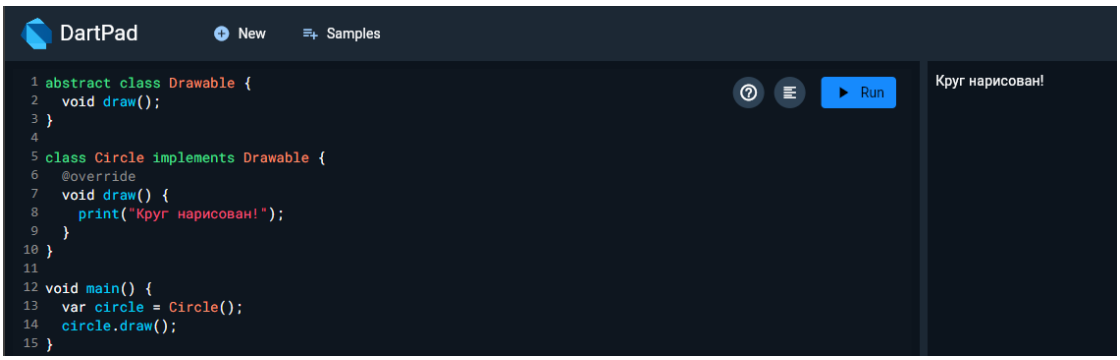


Рисунок №5 – Результат решения задания №5.

Задание № 6: Использование статических методов и свойств

Создайте класс *MathUtils* с статическим методом *calculateCircleArea*, который принимает радиус и возвращает площадь круга. Вызовите этот метод для расчета площади круга.

Описание алгоритма решения:

1. Определите класс *MathUtils*.
2. Добавьте в класс *MathUtils* статический метод *calculateCircleArea*, который принимает радиус и возвращает площадь круга.
3. Вызовите статический метод *calculateCircleArea* для расчета площади круга.

Рисунок №6 – Результат решения задания №6.

Задание № 7: Использование геттеров и сеттеров

Создайте класс *Person* с приватным свойством *_age* (целое число). Добавьте геттер и сеттер для этого свойства. Убедитесь, что возраст не может быть отрицательным.

Описание алгоритма решения:

1. Определите класс *Person*.
2. Добавьте в класс приватное свойство *_age*.
3. Добавьте геттер для свойства *_age*, который возвращает его значение.
4. Добавьте сеттер для свойства *_age*, который устанавливает его значение, если оно не отрицательное.
5. Создайте объект класса *Person* и установите ему возраст с помощью сеттера.
6. Выведите возраст с помощью геттера.

Рисунок №7 – Результат решения задания №7.

Задание № 8: Использование фабричных конструкторов

Создайте класс *Logger* с фабричным конструктором, который возвращает единственный экземпляр класса (*синглтон*). Добавьте метод *log*, который выводит сообщение.

Описание алгоритма решения:

1. Определите класс *Logger*.
2. Добавьте в класс приватное статическое свойство *_instance*, которое будет хранить единственный экземпляр класса.
3. Добавьте фабричный конструктор, который возвращает единственный экземпляр класса.
4. Добавьте метод *log*, который выводит сообщение.
5. Создайте объект класса *Logger* и вызовите его метод *log*.

Рисунок №8 – Результат решения задания №8.

Задание № 9: Использование перечислений (enums)

Создайте перечисление *DayOfWeek* с днями недели. Создайте функцию, которая принимает день недели и выводит сообщение о том, является ли этот день выходным.

Описание алгоритма решения:

1. Определите перечисление *DayOfWeek* с днями недели.
2. Создайте функцию, которая принимает день недели и проверяет, является ли он выходным (суббота или воскресенье).
3. Выведите соответствующее сообщение.

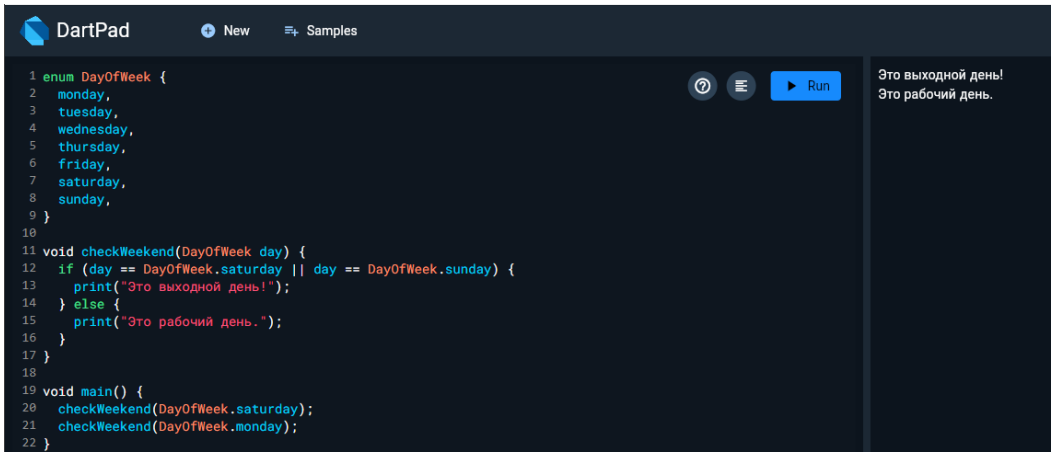


Рисунок №9 – Результат решения задания №9.

Задание № 10: Использование mixins

Создайте миксин *Flyable* с методом *fly*, который выводит сообщение о том, что объект может летать. Создайте класс *Bird*, который использует миксин *Flyable*.

Описание алгоритма решения:

1. Определите миксин Flyable с методом fly.
2. Определите класс Bird, который использует миксин Flyable.
3. Создайте объект класса Bird и вызовите его метод fly.

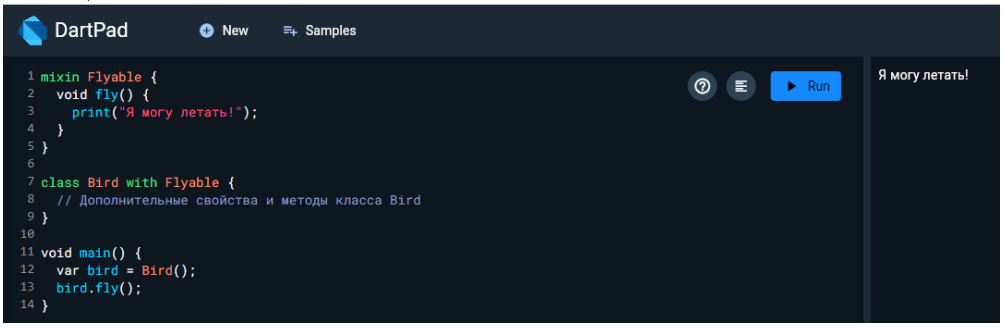


Рисунок №10 – Результат решения задания №10.

Задание № 11: Использование оператора is для проверки типа объекта

Создайте класс *Animal* и его подклассы *Dog* и *Cat*. Создайте список объектов типа *Animal* и проверьте, какие из них являются объектами класса *Dog*.

Описание алгоритма решения:

1. Определите класс *Animal*.
2. Определите классы *Dog* и *Cat*, которые наследуются от класса *Animal*.
3. Создайте список объектов типа *Animal*, содержащий объекты классов *Dog* и *Cat*.
4. Проверьте, какие из объектов являются объектами класса *Dog* с помощью оператора *is.*

Рисунок №11 – Результат решения задания №11.

Задание № 12: Использование оператора as для приведения типа объекта

Создайте класс *Vehicle* и его подкласс *Car*. Создайте объект типа *Vehicle* и приведите его к типу *Car*.

Описание алгоритма решения:

1. Определите класс *Vehicle*.
2. Определите класс *Car*, который наследуется от класса *Vehicle*.
3. Создайте объект типа *Vehicle*, который на самом деле является объектом класса *Car*.
4. Приведите объект к типу *Car* с помощью оператора *as*.

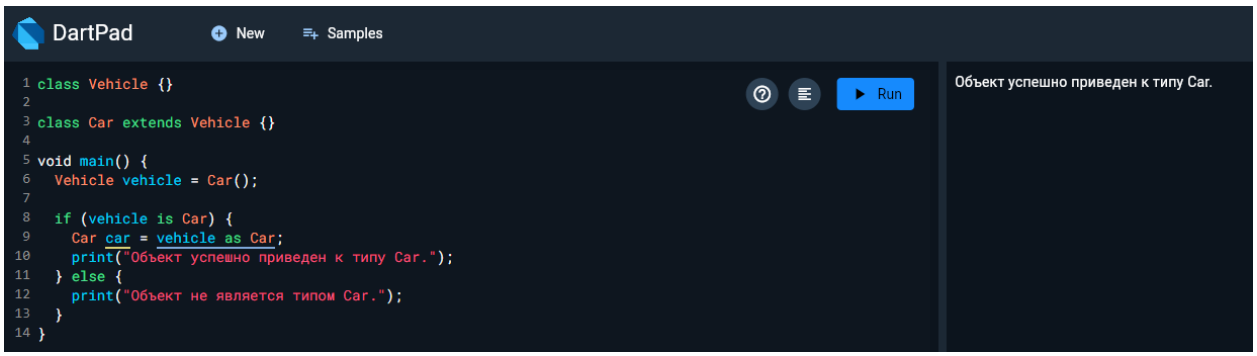


Рисунок №12 – Результат решения задания №12.

Задание № 13: Использование оператора ?? для проверки на null

Создайте класс *Person* с свойством name. Создайте объект класса *Person* и выведите его имя, используя оператор *??* для проверки на *null*.

Описание алгоритма решения:

1. Определите класс *Person* с свойством name.
2. Создайте объект класса *Person* с именем, равным *null*.
3. Выведите имя объекта, используя оператор *??* для проверки на *null* и предоставления значения по умолчанию.

Рисунок №13 – Результат решения задания №13.

Задание № 14: Использование оператора «?.» для безопасного доступа к свойствам

Создайте класс *Person* с свойством *address*, которое является объектом класса *Address*. Создайте объект класса *Person* с *address*, равным *null*, и выведите его город, используя оператор «?.».

Описание алгоритма решения:

1. Определите класс *Address* с свойством *city*.
2. Определите класс *Person* с свойством *address*, которое является объектом класса *Address*.
3. Создайте объект класса *Person* с *address*, равным *null*.
4. Выведите город объекта *address*, используя оператор «?.» для безопасного доступа.

Рисунок №14 – Результат решения задания №14.

Задание № 15: Использование оператора «..» (cascade notation).

Создайте класс *Person* с методами *setName* и *setAge*. Создайте объект класса *Person* и используйте оператор «..» для вызова нескольких методов подряд.

Описание алгоритма решения:

1. Определите класс *Person* с методами *setName* и *setAge*.
2. Создайте объект класса *Person*.
3. Используйте оператор «..» для вызова нескольких методов подряд.

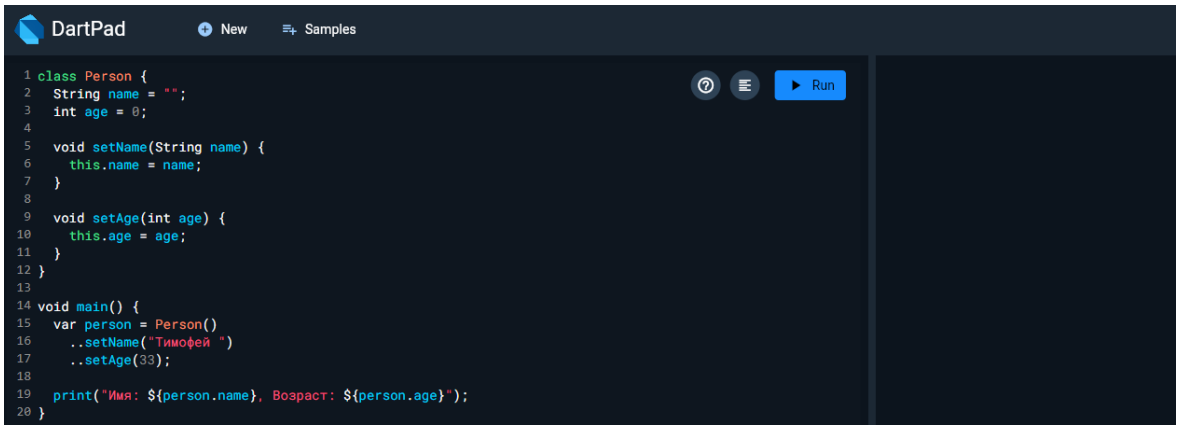


Рисунок №15 – Результат решения задания №15.

Задание № 16: Использование оператора late

Создайте класс *Person* с отложенным свойством *address*, которое будет инициализировано позже.

Описание алгоритма решения:

1. Определите класс *Person* с отложенным свойством *address*, используя оператор *late*.
2. Создайте объект класса *Person* и инициализируйте свойство *address* позже.

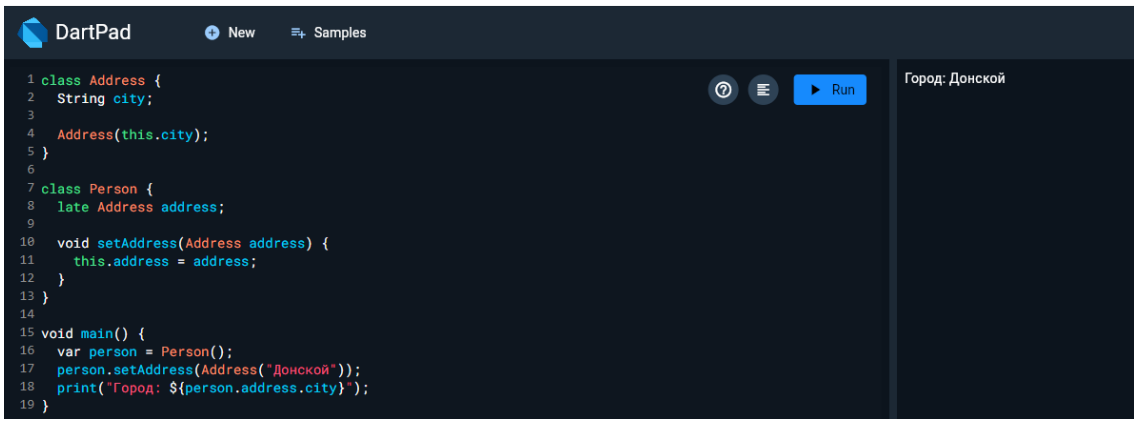


Рисунок №16 – Результат решения задания №16.

Задание № 17: Использование оператора required в конструкторе.

Создайте класс *Person* с конструктором, который требует обязательные параметры *name* и *age*.

Описание алгоритма решения:

1. Определите класс *Person* с конструктором, который использует оператор *required* для обязательных параметров *name* и *age*.
2. Создайте объект класса *Person*, передав обязательные параметры.

Рисунок №17 – Результат решения задания №17.

Задание № 18: Использование оператора const для создания неизменяемых объектов

Создайте класс *Point* с неизменяемыми свойствами *x и y*. Создайте константный объект класса *Point*.

Описание алгоритма решения:

1. Определите класс *Point* с неизменяемыми свойствами *x и y*.
2. Создайте константный объект класса *Point*, используя оператор *const*.

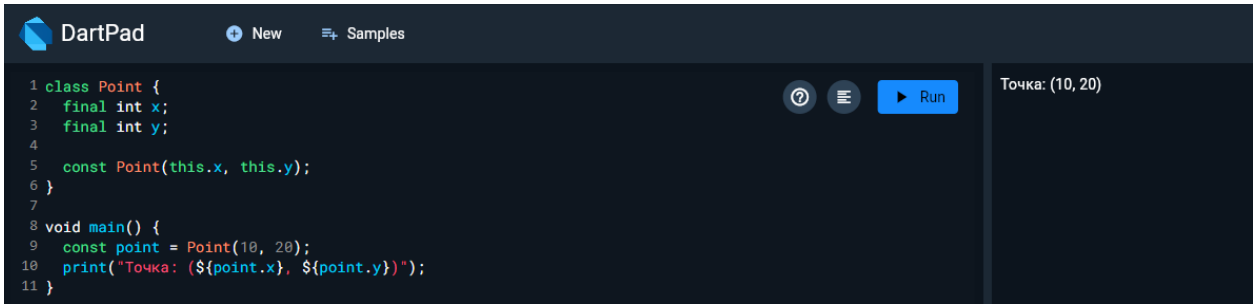


Рисунок №18 – Результат решения задания №18.

Задание № 19: Использование оператора final для создания неизменяемых свойств

Создайте класс *Circle* с неизменяемым свойством *radius*. Создайте объект класса *Circle* и выведите его радиус.

Описание алгоритма решения:

1. Определите класс *Circle* с неизменяемым свойством *radius*, используя оператор *final*.
2. Создайте объект класса *Circle* и выведите его радиус.

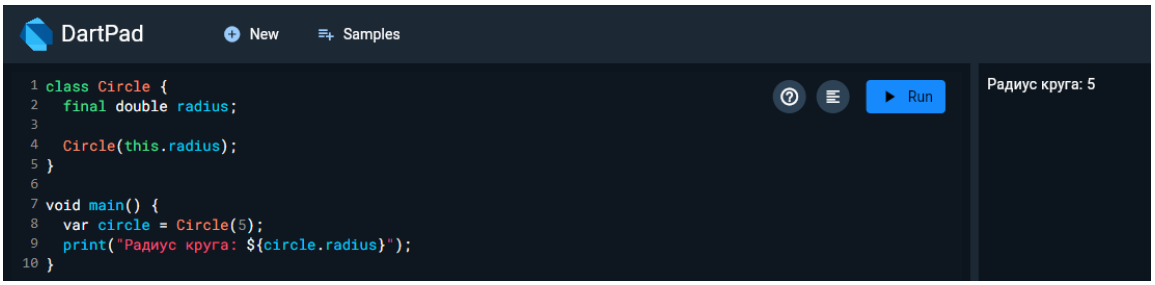


Рисунок №19 – Результат решения задания №19.

Задание № 20: Использование оператора assert для проверки условий

Создайте класс *Person* с конструктором, который проверяет, что возраст не отрицательный, используя оператор *assert*.

Описание алгоритма решения:

1. Определите класс *Person* с конструктором, который принимает параметр *age*.
2. Используйте оператор *assert* для проверки, что возраст не отрицательный.
3. Создайте объект класса *Person* и передайте ему корректный и некорректный возраст.

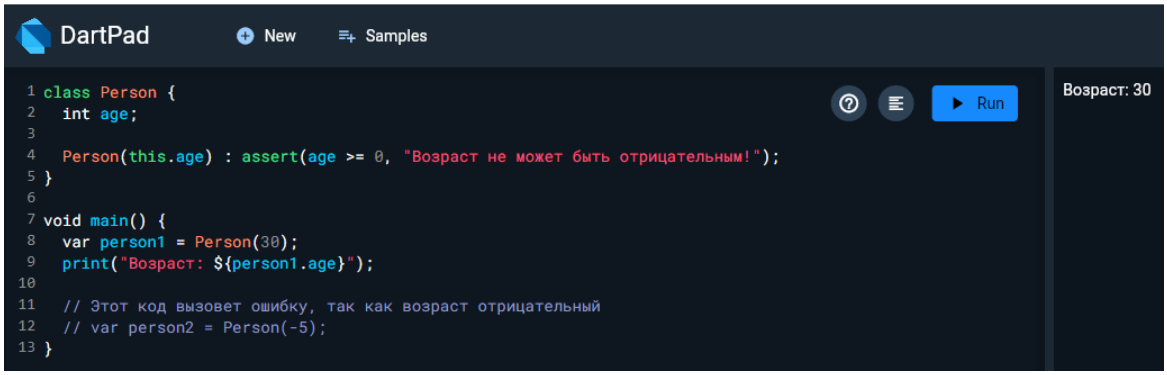


Рисунок №20 – Результат решения задания №20.

Индивидуальные задания для закрепления материала

1. **Простой калькулятор:** Создайте класс Calculator, который будет содержать методы для выполнения основных арифметических операций (сложение, вычитание, умножение, деление). Создайте экземпляр класса и используйте его для вычисления выражения, введенного пользователем.
2. **Книжная полка:** Создайте класс Book, который будет содержать информацию о книге (название, автор, год издания). Создайте класс Bookshelf, который будет хранить список книг. Реализуйте методы для добавления, удаления и вывода списка книг на экран.
3. ***Список задач:*** Создайте класс Task, который будет содержать информацию о задаче (название, описание, статус выполнения). Создайте класс TaskList, который будет хранить список задач. Реализуйте методы для добавления, удаления, изменения статуса и вывода списка задач на экран.
4. **Банковский счет:** Создайте класс BankAccount, который будет содержать информацию о банковском счете (номер счета, баланс, владелец). Реализуйте методы для пополнения счета, снятия денег и проверки баланса.
5. **Игра «Угадай число»:** Создайте класс NumberGuesser, который будет генерировать случайное число и проверять предположения пользователя. Реализуйте методы для получения предположения пользователя и вывода результата (больше, меньше, угадал).
6. **Конвертер валют:** Создайте класс CurrencyConverter, который будет содержать курсы обмена валют. Реализуйте метод для конвертации суммы из одной валюты в другую.
7. **Генератор паролей:** Создайте класс PasswordGenerator, который будет генерировать случайные пароли заданной длины. Добавьте возможность указания используемых символов (буквы, цифры, специальные символы).
8. **Простой калькулятор дробей:** Создайте класс Fraction, который будет представлять дробь (числитель и знаменатель). Реализуйте методы для сложения, вычитания, умножения и деления дробей.
9. **Игра «Крестики-нолики»:** Создайте класс TicTacToe, который будет реализовывать игру "Крестики-нолики". Класс должен содержать игровое поле и методы для хода игроков и определения победителя.
10. **Конвертер температур:** Создайте класс TemperatureConverter, который будет конвертировать температуру между градусами Цельсия, Фаренгейта и Кельвина.

11. **Игра «Быки и коровы»:** Создайте класс BullsAndCows, который будет реализовывать игру «Быки и коровы». Класс должен генерировать случайное число и проверять предположения пользователя.
12. **Простой калькулятор комплексных чисел:** Создайте класс ComplexNumber, который будет представлять комплексное число (действительная и мнимая части). Реализуйте методы для сложения, вычитания, умножения и деления комплексных чисел.
13. **Конвертер единиц измерения:** Создайте класс UnitConverter, который будет конвертировать различные единицы измерения (длина, вес, объем).
14. **Простой калькулятор векторов:** Создайте класс Vector, который будет представлять вектор (координаты x, y, z). Реализуйте методы для сложения, вычитания, скалярного и векторного произведения векторов.
15. **Конвертер римских чисел:** Создайте класс RomanNumeralConverter, который будет конвертировать римские числа в арабские и наоборот.

Дополнительные рекомендации:

1. Используйте комментарии для пояснения работы классов.
2. Проверяйте корректность ввода данных пользователем.
3. Оформляйте код в соответствии с общепринятыми стандартами.
4. Тестируйте программу на различных наборах данных.
5. Представить результат работы в рукописном виде в рабочей тетради и в электронном формате с использованием Git-репозитория с предоставление QR-кода на репозиторий.

Критерии оценивания:

«Отлично» - выполнены все тренировочные задания и девять индивидуальных заданий.

«Хорошо» - выполнены все тренировочные задания и шесть индивидуальных заданий.

«Удовлетворительно» - выполнены все тренировочные задания и три индивидуальных задания.