

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().



```
1 class Person {
2   String name;
3   int age;
4
5   Person(this.name, this.age);
6 }
7
8 void main() {
9   var person = Person("Тимофей", 33);
10  print("Имя: ${person.name}\nВозраст: ${person.age}");
11 }
```

Имя: Тимофей
Возраст: 33

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.



```
1 class Person {
2   String name;
3   int age;
4
5   Person(this.name, this.age);
6
7   void sayHello() {
8     print("Привет, меня зовут ${this.name}!");
9   }
10 }
11
12 void main() {
13   var person = Person("Тимофей", 33);
14   person.sayHello();
15 }
```

Привет, меня зовут Тимофей!

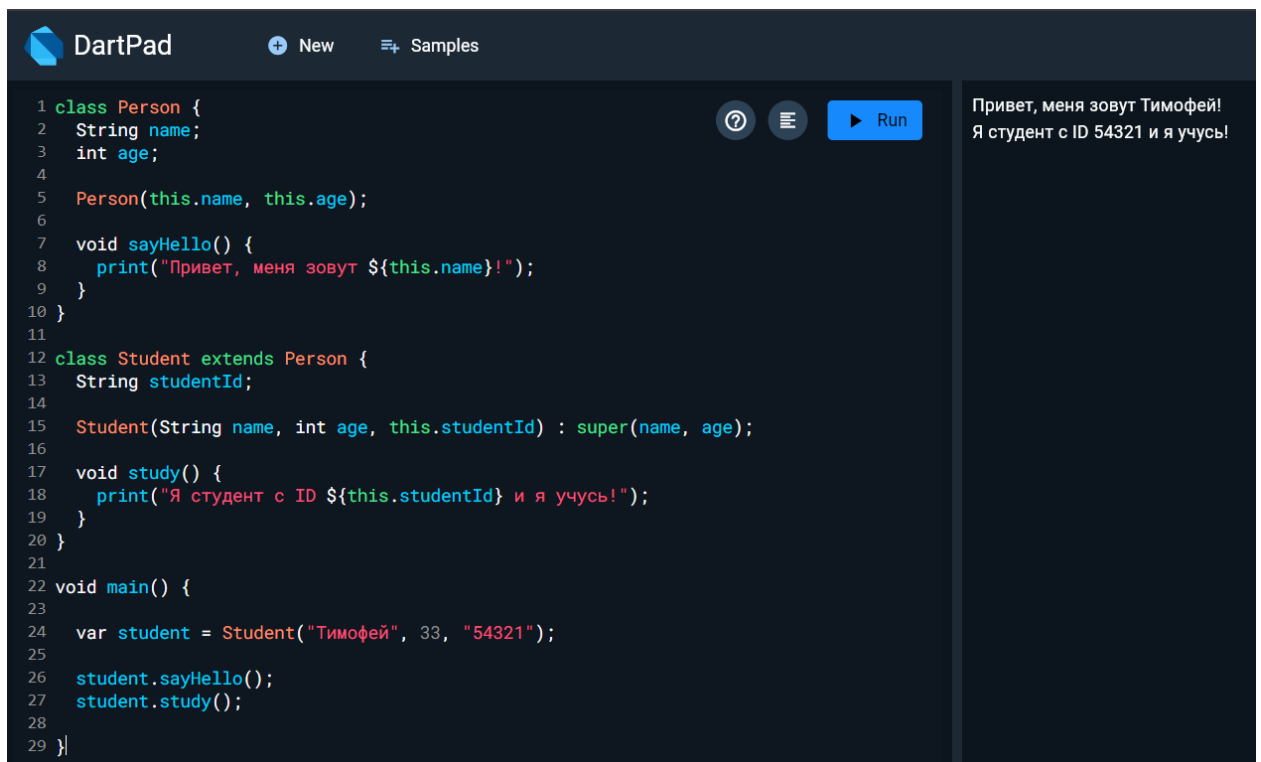
Рисунок №2 – Результат решения задания №2.

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*.



```
1 class Person {
2   String name;
3   int age;
4
5   Person(this.name, this.age);
6
7   void sayHello() {
8     print("Привет, меня зовут ${this.name}!");
9   }
10 }
11
12 class Student extends Person {
13   String studentId;
14
15   Student(String name, int age, this.studentId) : super(name, age);
16
17   void study() {
18     print("Я студент с ID ${this.studentId} и я учусь!");
19   }
20 }
21
22 void main() {
23
24   var student = Student("Тимофей", 33, "54321");
25
26   student.sayHello();
27   student.study();
28
29 }
```

Привет, меня зовут Тимофей!
Я студент с ID 54321 и я учусь!

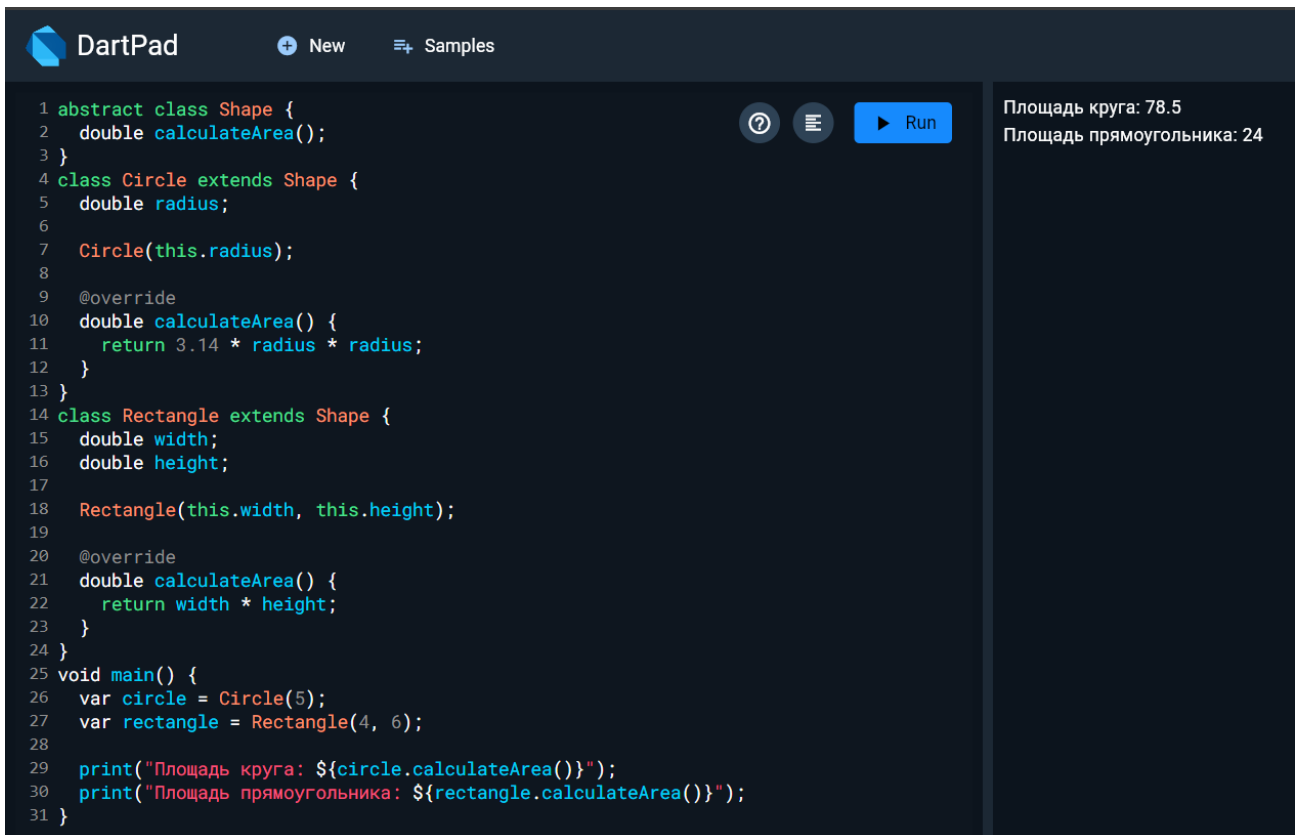
Рисунок №3 – Результат решения задания №3.

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



```
1 abstract class Shape {
2   double calculateArea();
3 }
4 class Circle extends Shape {
5   double radius;
6
7   Circle(this.radius);
8
9   @override
10  double calculateArea() {
11    return 3.14 * radius * radius;
12  }
13 }
14 class Rectangle extends Shape {
15   double width;
16   double height;
17
18   Rectangle(this.width, this.height);
19
20   @override
21   double calculateArea() {
22     return width * height;
23   }
24 }
25 void main() {
26   var circle = Circle(5);
27   var rectangle = Rectangle(4, 6);
28
29   print("Площадь круга: ${circle.calculateArea()}");
30   print("Площадь прямоугольника: ${rectangle.calculateArea()}");
31 }
```

Площадь круга: 78.5
Площадь прямоугольника: 24

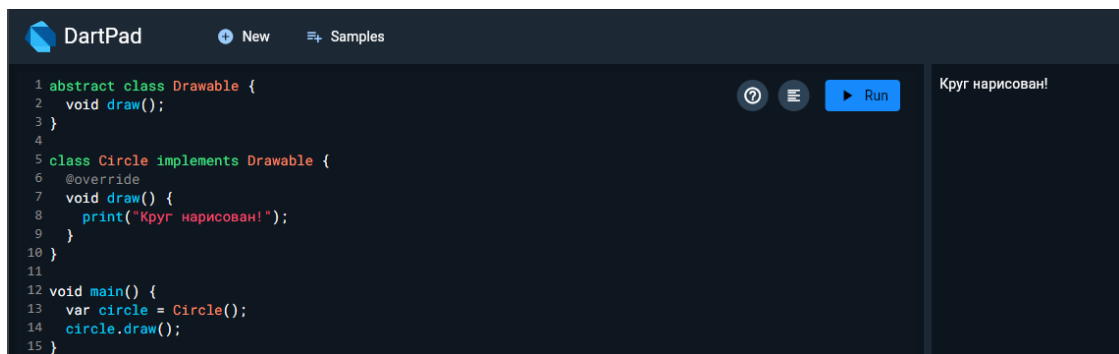
Рисунок №4 – Результат решения задания №4.

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



```
1 abstract class Drawable {
2   void draw();
3 }
4
5 class Circle implements Drawable {
6   @override
7   void draw() {
8     print("Круг нарисован!");
9   }
10 }
11
12 void main() {
13   var circle = Circle();
14   circle.draw();
15 }
```

Круг нарисован!

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.



```
1 class MathUtils {
2   static double calculateCircleArea(double radius) {
3     return 3.14 * radius * radius;
4   }
5 }
6
7 void main() {
8   double radius = 5;
9   double area = MathUtils.calculateCircleArea(radius);
10  print("Площадь круга с радиусом $radius: $area");
11 }
```

Площадь круга с радиусом 5: 78.5

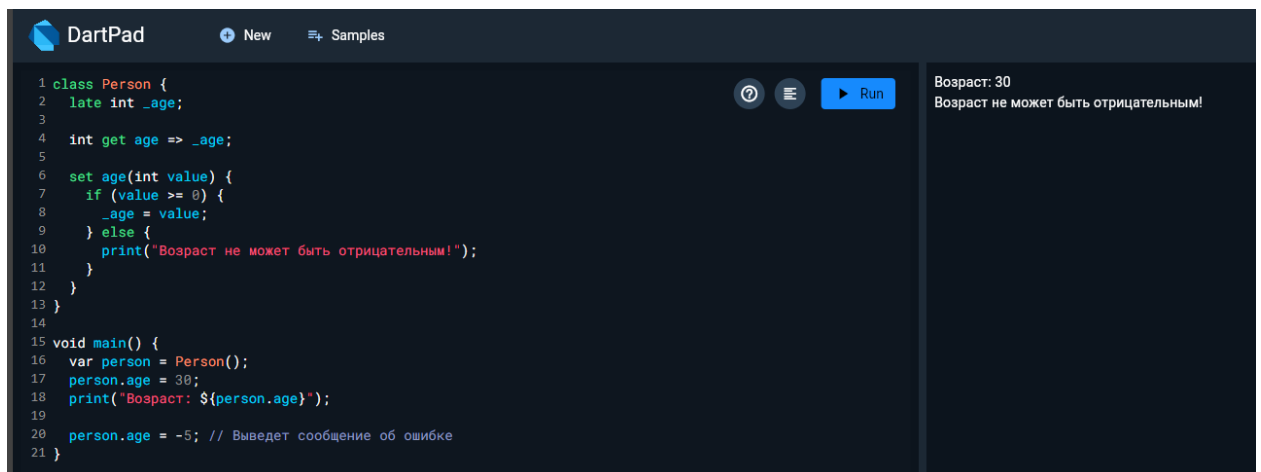
Рисунок №6 – Результат решения задания №6.

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.



```
1 class Person {
2   late int _age;
3
4   int get age => _age;
5
6   set age(int value) {
7     if (value >= 0) {
8       _age = value;
9     } else {
10      print("Возраст не может быть отрицательным!");
11    }
12  }
13 }
14
15 void main() {
16   var person = Person();
17   person.age = 30;
18   print("Возраст: ${person.age}");
19
20   person.age = -5; // Выведет сообщение об ошибке
21 }
```

Возраст: 30
Возраст не может быть отрицательным!

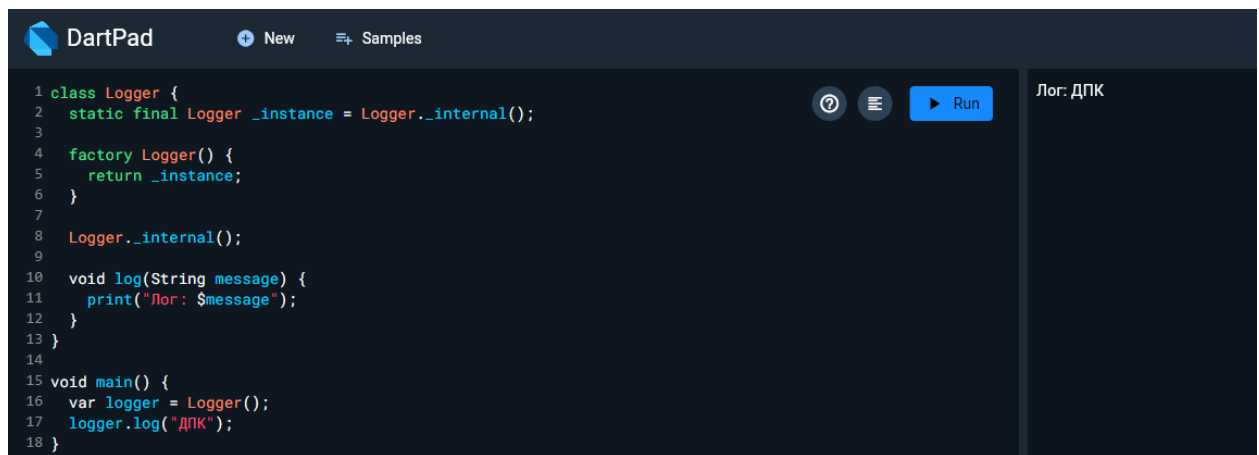
Рисунок №7 – Результат решения задания №7.

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.



```
1 class Logger {
2   static final Logger _instance = Logger._internal();
3
4   factory Logger() {
5     return _instance;
6   }
7
8   Logger._internal();
9
10  void log(String message) {
11    print("Лог: $message");
12  }
13 }
14
15 void main() {
16   var logger = Logger();
17   logger.log("ДПК");
18 }
```

Лог: ДПК

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.



```
1 enum DayOfWeek {
2   monday,
3   tuesday,
4   wednesday,
5   thursday,
6   friday,
7   saturday,
8   sunday,
9 }
10
11 void checkWeekend(DayOfWeek day) {
12   if (day == DayOfWeek.saturday || day == DayOfWeek.sunday) {
13     print("Это выходной день!");
14   } else {
15     print("Это рабочий день.");
16   }
17 }
18
19 void main() {
20   checkWeekend(DayOfWeek.saturday);
21   checkWeekend(DayOfWeek.monday);
22 }
```

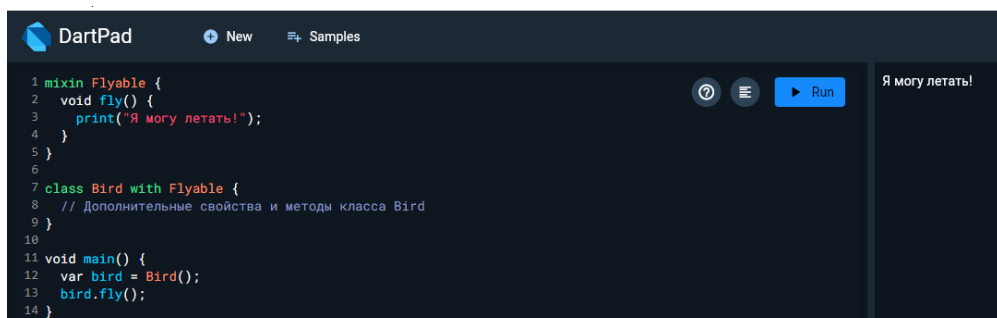
Рисунок №9 – Результат решения задания №9.

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.



```
1 mixin Flyable {
2   void fly() {
3     print("Я могу летать!");
4   }
5 }
6
7 class Bird with Flyable {
8   // Дополнительные свойства и методы класса Bird
9 }
10
11 void main() {
12   var bird = Bird();
13   bird.fly();
14 }
```

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*.



```
1 class Animal {}
2
3 class Dog extends Animal {}
4
5 class Cat extends Animal {}
6
7 void main() {
8   var animals = [Dog(), Cat(), Dog(), Cat()];
9
10  for (var animal in animals) {
11    if (animal is Dog) {
12      print("Это собака!");
13    } else if (animal is Cat) {
14      print("Это кошка.");
15    }
16  }
17 }
```

Это собака!
Это кошка.
Это собака!
Это кошка.

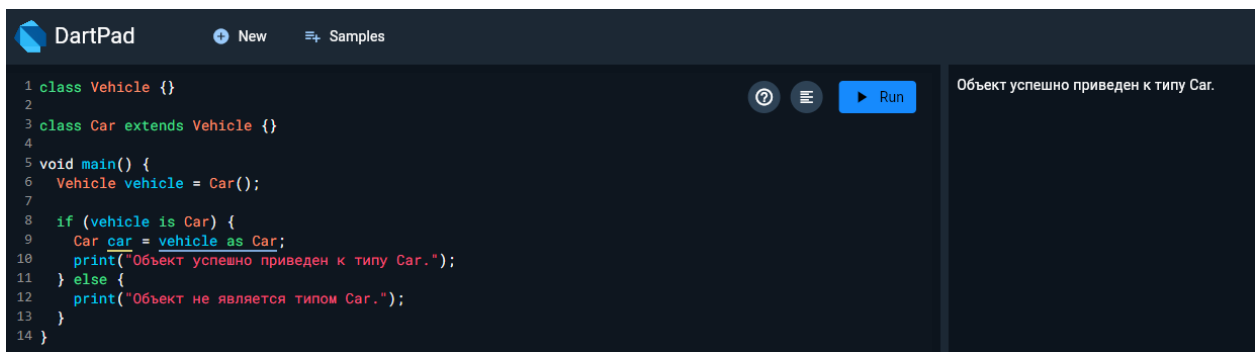
Рисунок №11 – Результат решения задания №11.

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`.



```
1 class Vehicle {}
2
3 class Car extends Vehicle {}
4
5 void main() {
6   Vehicle vehicle = Car();
7
8   if (vehicle is Car) {
9     Car car = vehicle as Car;
10    print("Объект успешно приведен к типу Car.");
11  } else {
12    print("Объект не является типом Car.");
13  }
14 }
```

Объект успешно приведен к типу Car.

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.



```
1 class Person {
2   String? name;
3
4   Person(this.name);
5 }
6
7 void main() {
8   var person = Person(null);
9   print("Имя: ${person.name ?? "Неизвестно"}");
10 }
```

Имя: Неизвестно

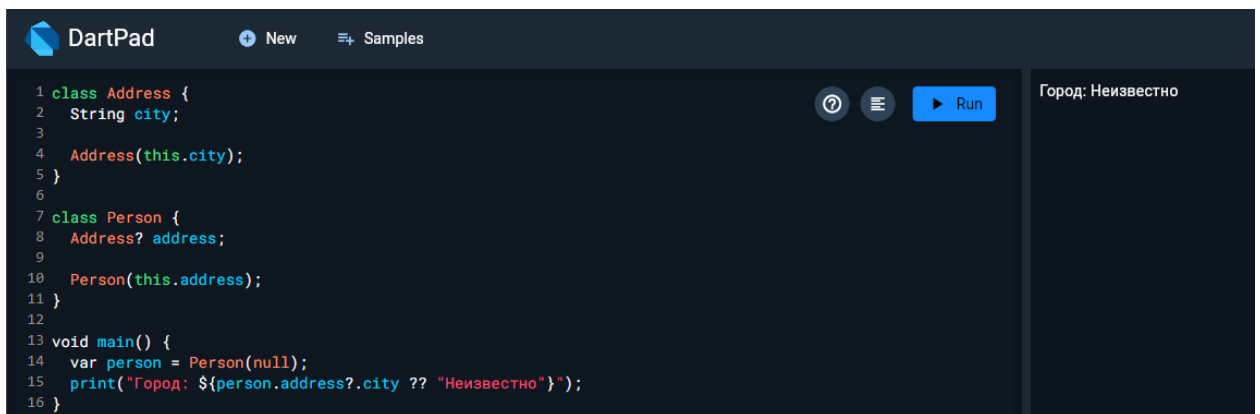
Рисунок №13 – Результат решения задания №13.

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.



```
1 class Address {
2   String city;
3
4   Address(this.city);
5 }
6
7 class Person {
8   Address? address;
9
10  Person(this.address);
11 }
12
13 void main() {
14   var person = Person(null);
15   print("Город: ${person.address?.city ?? "Неизвестно"}");
16 }
```

Город: Неизвестно

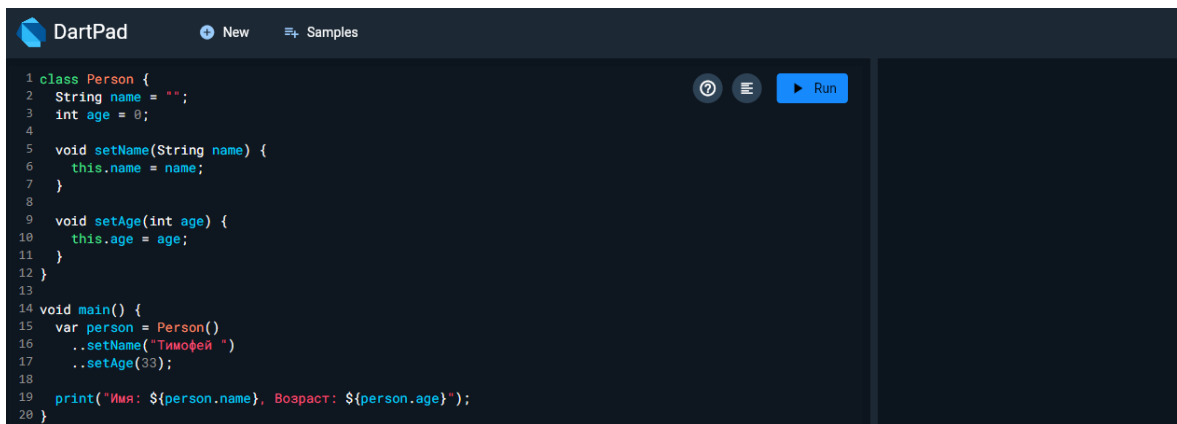
Рисунок №14 – Результат решения задания №14.

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.



```
1 class Person {
2   String name = "";
3   int age = 0;
4
5   void setName(String name) {
6     this.name = name;
7   }
8
9   void setAge(int age) {
10    this.age = age;
11  }
12 }
13
14 void main() {
15   var person = Person()
16   ..setName("Тимофеев")
17   ..setAge(33);
18
19   print("Имя: ${person.name}, Возраст: ${person.age}");
20 }
```

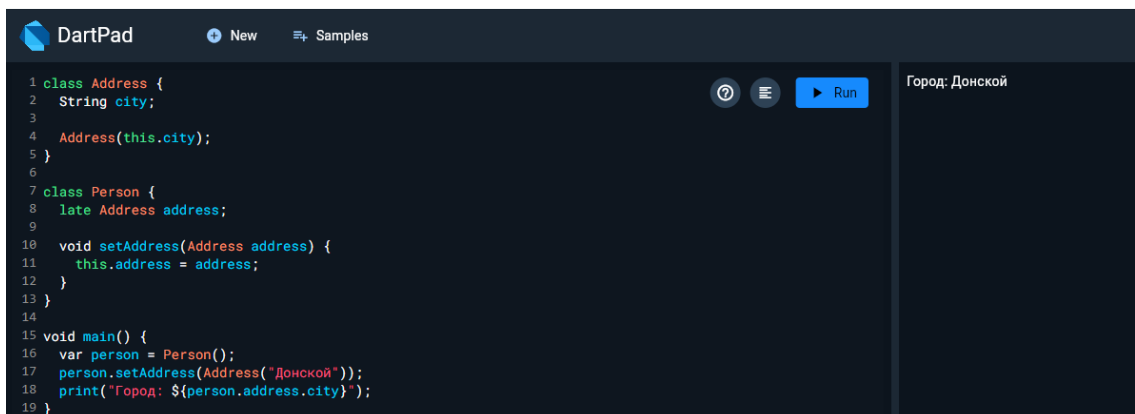
Рисунок №15 – Результат решения задания №15.

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.



```
1 class Address {
2   String city;
3
4   Address(this.city);
5 }
6
7 class Person {
8   late Address address;
9
10  void setAddress(Address address) {
11    this.address = address;
12  }
13 }
14
15 void main() {
16   var person = Person();
17   person.setAddress(Address("Донской"));
18   print("Город: ${person.address.city}");
19 }
```


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.



The screenshot shows the DartPad interface. On the left, the code editor contains the following Dart code:

```
1 class Person {  
2   String name;  
3   int age;  
4  
5   Person({required this.name, required this.age});  
6 }  
7  
8 void main() {  
9   var person = Person(name: "Тимофей", age: 33);  
10  print("Имя: ${person.name}, Возраст: ${person.age}");  
11 }
```

On the right, the output console displays the result of running the code: "Имя: Тимофей, Возраст: 33".

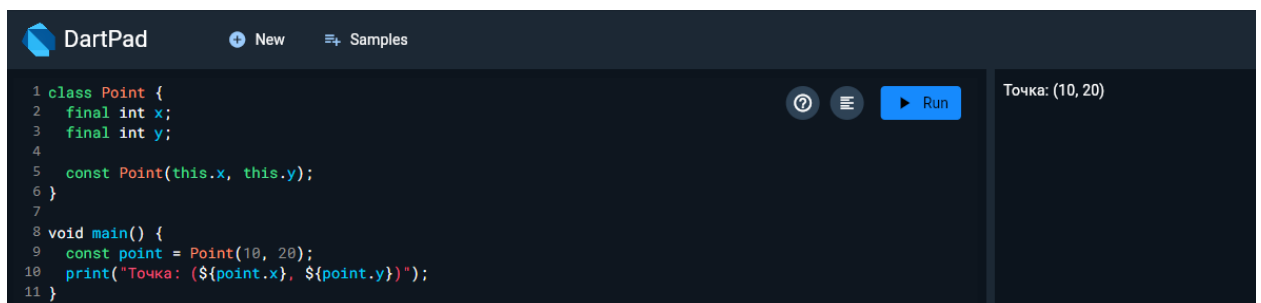
Рисунок №17 – Результат решения задания №17.

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*.



The screenshot shows the DartPad interface. On the left, the code editor contains the following Dart code:

```
1 class Point {  
2   final int x;  
3   final int y;  
4  
5   const Point(this.x, this.y);  
6 }  
7  
8 void main() {  
9   const point = Point(10, 20);  
10  print("Точка: (${point.x}, ${point.y})");  
11 }
```

On the right, the output console displays the result of running the code: "Точка: (10, 20)".

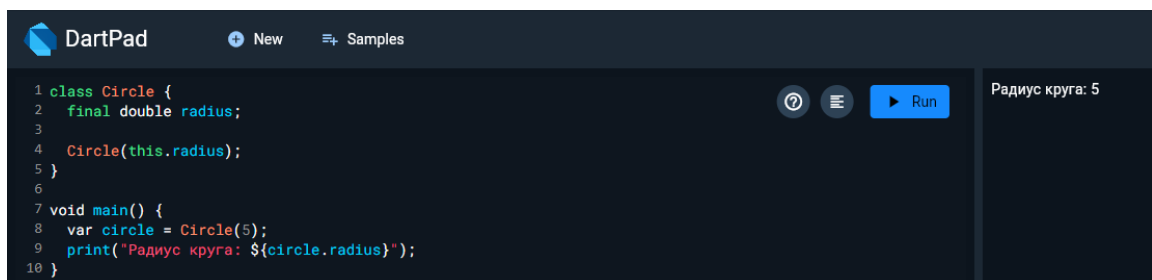
Рисунок №18 – Результат решения задания №18.

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.



```
1 class Circle {
2   final double radius;
3
4   Circle(this.radius);
5 }
6
7 void main() {
8   var circle = Circle(5);
9   print("Радиус круга: ${circle.radius}");
10 }
```

Радиус круга: 5

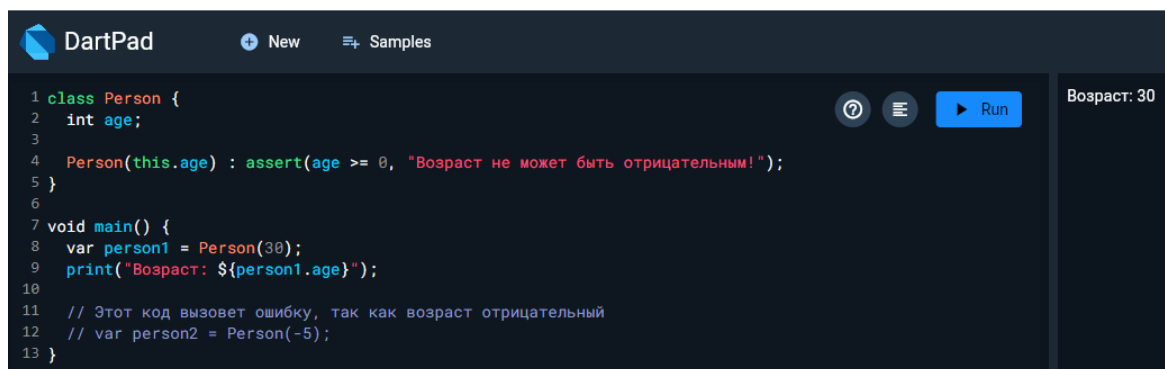
Рисунок №19 – Результат решения задания №19.

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.



```
1 class Person {
2   int age;
3
4   Person(this.age) : assert(age >= 0, "Возраст не может быть отрицательным!");
5 }
6
7 void main() {
8   var person1 = Person(30);
9   print("Возраст: ${person1.age}");
10
11   // Этот код вызовет ошибку, так как возраст отрицательный
12   // var person2 = Person(-5);
13 }
```

Возраст: 30

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.