

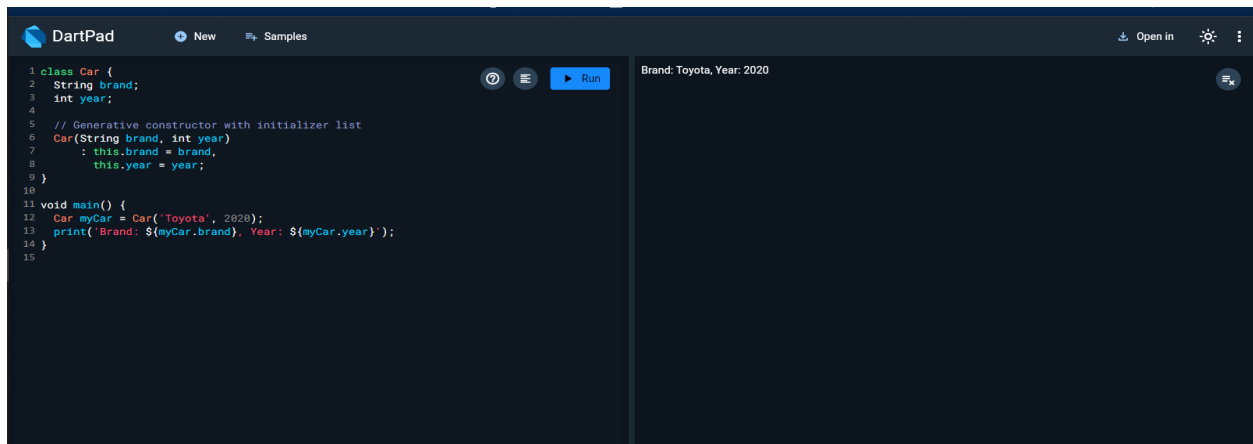
Assignment # 4: Types of Constructors in Dart

By: Durr e Najaf

1. Generative Constructors

A generative constructor is the most common type of constructor in Dart. It's used to create an instance of a class. If you don't provide any constructors in a class, Dart creates a default generative constructor.

Example:



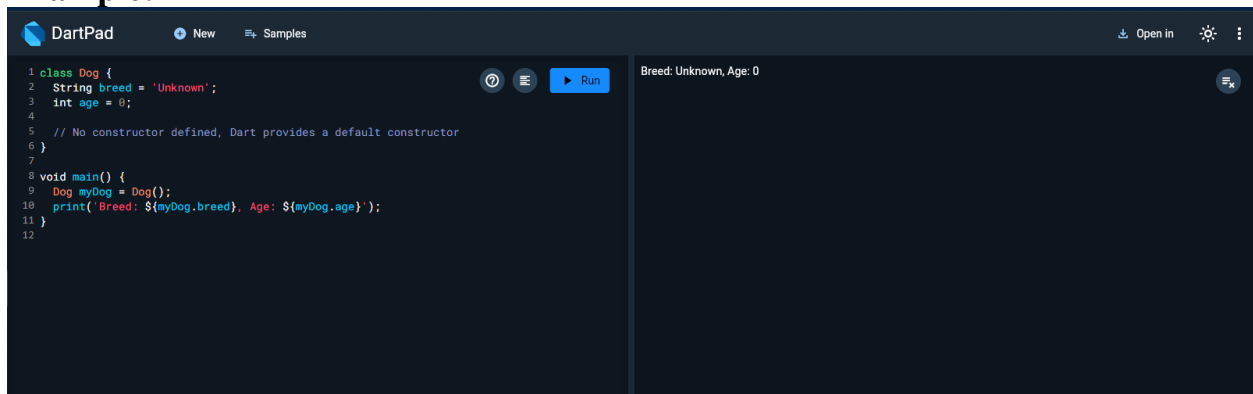
```
1 class Car {
2   String brand;
3   int year;
4
5   // Generative constructor with initializer list
6   Car(String brand, int year)
7     : this.brand = brand,
8       this.year = year;
9 }
10
11 void main() {
12   Car myCar = Car('Toyota', 2020);
13   print('Brand: ${myCar.brand}, Year: ${myCar.year}');
14 }
15
```

Brand: Toyota, Year: 2020

2. Default Constructors

A default constructor is a type of generative constructor that Dart provides automatically if you don't define any constructors in your class. It has no parameters and simply initializes the object.

Example:



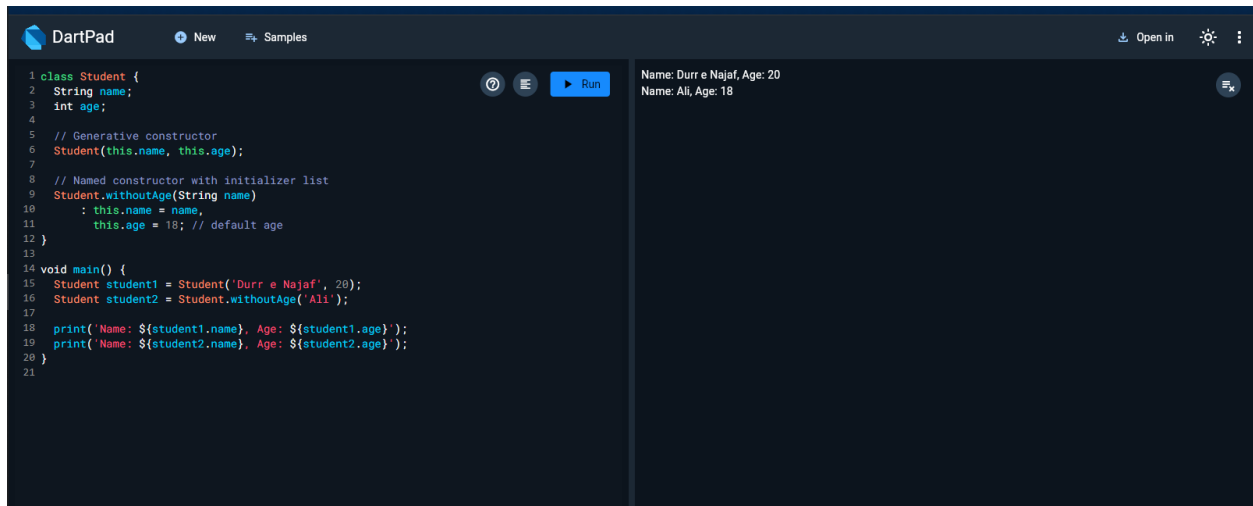
```
1 class Dog {
2   String breed = 'Unknown';
3   int age = 0;
4
5   // No constructor defined, Dart provides a default constructor
6 }
7
8 void main() {
9   Dog myDog = Dog();
10  print('Breed: ${myDog.breed}, Age: ${myDog.age}');
11 }
12
```

Breed: Unknown, Age: 0

3. Named Constructors

Named constructors allow you to create multiple constructors with different names in a class. This is useful when you want to create objects in different ways.

Example:



```
1 class Student {
2   String name;
3   int age;
4
5   // Generative constructor
6   Student(this.name, this.age);
7
8   // Named constructor with initializer list
9   Student.withoutAge(String name)
10    : this.name = name,
11      this.age = 18; // default age
12 }
13
14 void main() {
15   Student student1 = Student('Durr e Najaf', 20);
16   Student student2 = Student.withoutAge('Ali');
17
18   print('Name: ${student1.name}, Age: ${student1.age}');
19   print('Name: ${student2.name}, Age: ${student2.age}');
20 }
21
```

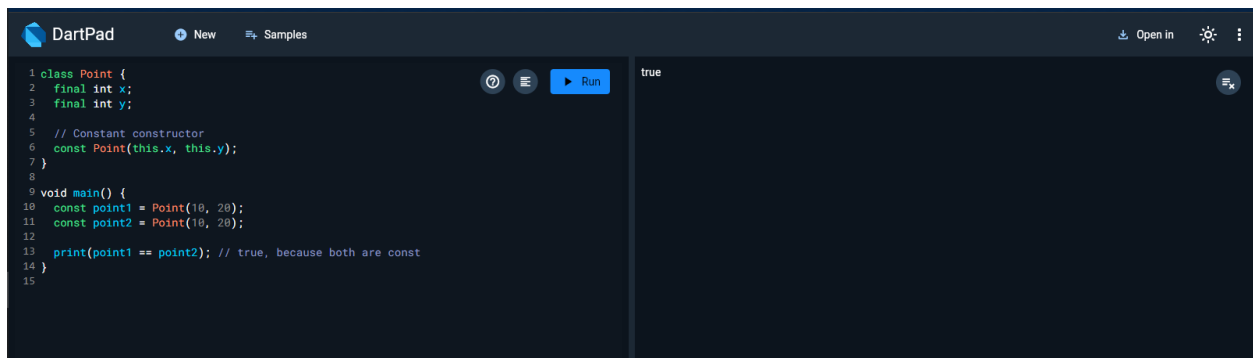
Output:

```
Name: Durr e Najaf, Age: 20
Name: Ali, Age: 18
```

4. Constant Constructors

Constant constructors are used when you want to create immutable objects (objects that can't be changed after creation). These objects must be marked as `const`.

Example:



```
1 class Point {
2   final int x;
3   final int y;
4
5   // Constant constructor
6   const Point(this.x, this.y);
7 }
8
9 void main() {
10   const point1 = Point(10, 20);
11   const point2 = Point(10, 20);
12
13   print(point1 == point2); // true, because both are const
14 }
15
```

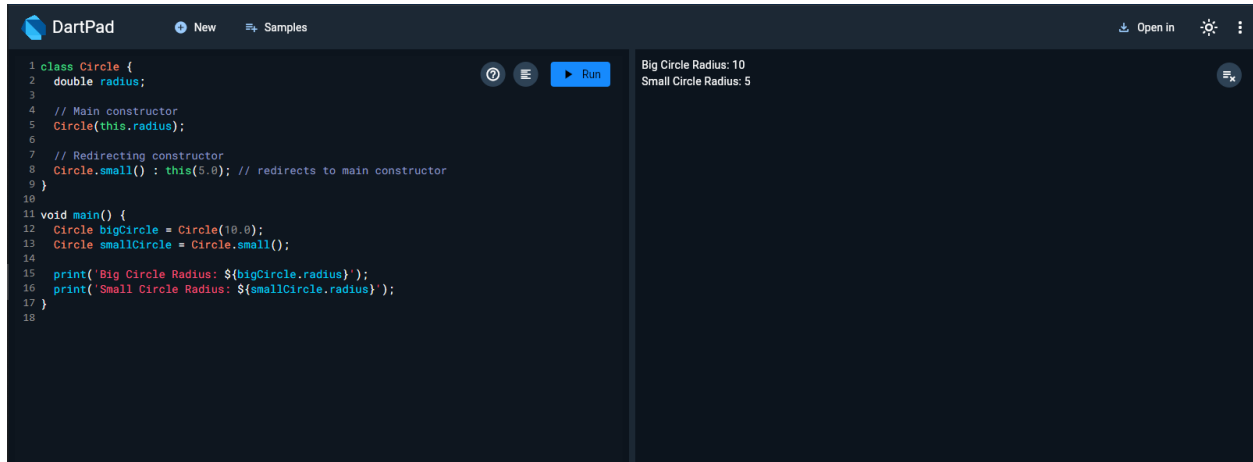
Output:

```
true
```

5. Redirecting Constructors

A redirecting constructor is used when you want one constructor to call another constructor in the same class. This helps to avoid code duplication.

Example:



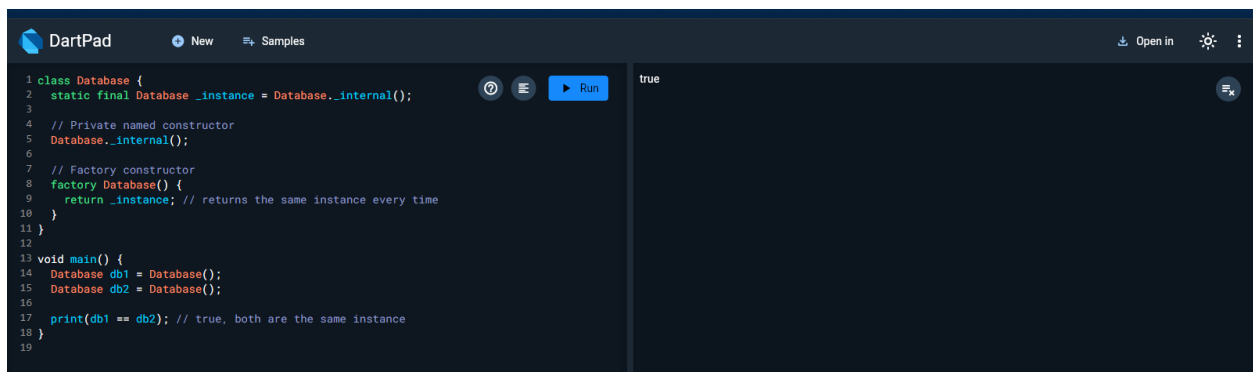
```
1 class Circle {
2   double radius;
3
4   // Main constructor
5   Circle(this.radius);
6
7   // Redirecting constructor
8   Circle.small() : this(5.0); // redirects to main constructor
9 }
10
11 void main() {
12   Circle bigCircle = Circle(10.0);
13   Circle smallCircle = Circle.small();
14
15   print('Big Circle Radius: ${bigCircle.radius}');
16   print('Small Circle Radius: ${smallCircle.radius}');
17 }
18
```

Big Circle Radius: 10
Small Circle Radius: 5

6. Factory Constructors

Factory constructors are used to return instances of a class. They can be used to control the object creation process, for example, by returning a cached object instead of creating a new one every time.

Example:



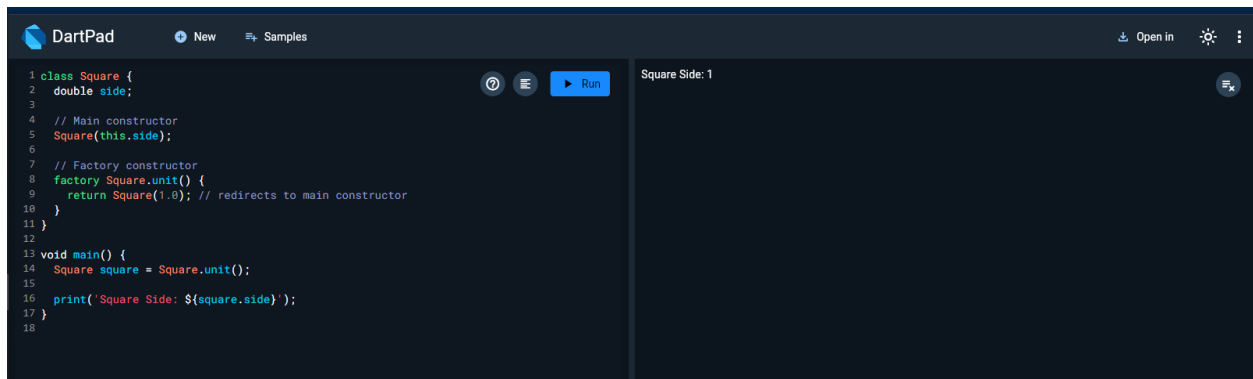
```
1 class Database {
2   static final Database _instance = Database._internal();
3
4   // Private named constructor
5   Database._internal();
6
7   // Factory constructor
8   factory Database() {
9     return _instance; // returns the same instance every time
10  }
11 }
12
13 void main() {
14   Database db1 = Database();
15   Database db2 = Database();
16
17   print(db1 == db2); // true, both are the same instance
18 }
19
```

true

7. Redirecting Factory Constructors

A redirecting factory constructor is a factory constructor that redirects to another constructor, either in the same class or in a different class.

Example:



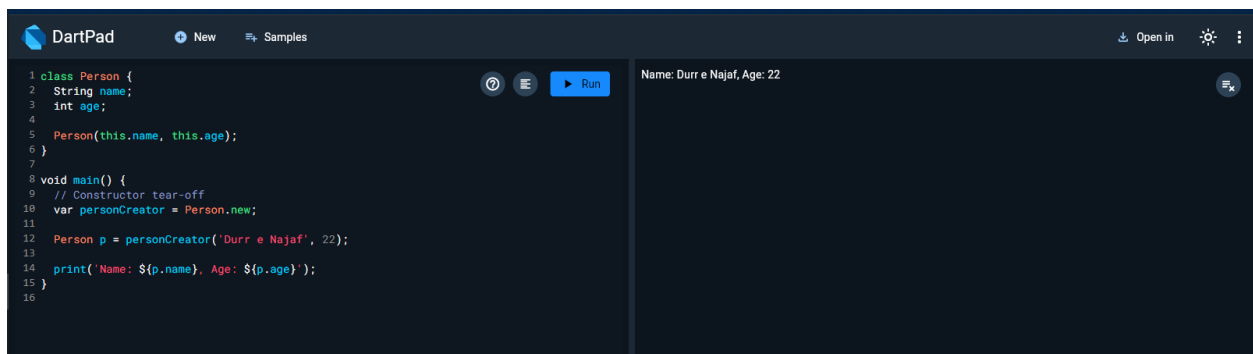
```
1 class Square {
2   double side;
3
4   // Main constructor
5   Square(this.side);
6
7   // Factory constructor
8   factory Square.unit() {
9     return Square(1.0); // redirects to main constructor
10  }
11 }
12
13 void main() {
14   Square square = Square.unit();
15
16   print('Square Side: ${square.side}');
17 }
18
```

Square Side: 1

8. Constructor Tear-offs

Constructor tear-offs allow you to pass a constructor as a function. This is useful when you want to create instances of a class dynamically or when you want to pass a constructor to a function.

Example:



```
1 class Person {
2   String name;
3   int age;
4
5   Person(this.name, this.age);
6 }
7
8 void main() {
9   // Constructor tear-off
10  var personCreator = Person.new;
11
12  Person p = personCreator('Durr e Najaf', 22);
13
14  print('Name: ${p.name}, Age: ${p.age}');
15 }
16
```

Name: Durr e Najaf, Age: 22