Для всех задач необходимо дополнительно создавать класс Application, который будет содержать статический метод main.

**Задача 1**

Создать класс Factory со статическим методом Car make(). Класс служит реализацией завода и производит машину при вызове метода make(). Тип машины выбирается случайно из таблицы ниже. Создать абстрактный класс Car без полей и методов. Создать абстрактные классы Regular и Premium. Создать реализации Renault, Nissan, Mitsubishi, Mercedes. Метод make() необходимо реализовывать по паттерну фабрики.

|  |  |  |
| --- | --- | --- |
| Марка | Тип | Шанс |
| Renault | Regular | 40% |
| Nissan | Regular | 30% |
| Mitsubishi | Regular | 20% |
| Mercedes | Premium | 10% |

В методе main необходимо создать список машин. Заполнить его 1000 полученными с помощью метода make() автомобилями. После заполнения необходимо вывести на экран строки, содержащие: Количество машин каждой марки, количество машин в каждом сегменте (Regular / Premium).

**Задача 2**

В данной задаче нам необходимо реализовать 2 интерфейса: Drawable и Movable. Интерфейс Drawable должен содержать метод draw(), интерфейс Movable – метод move().

Создать классы Square, Circle (квадрат, круг), которые должны реализовывать интерфейс Drawable и при вызове метода выводить на экран сообщение «Фигура <Тип фигуры> нарисована»

Создать классы Box, Table (ящик, стол), которые должны реализовывать интерфейс Movable и при вызове метода выводить на экран сообщение «Предмет <предмет> перемещен»

Созданные классы должны реализовывать только один соответствующий им интерфейс.

Создать класс Generator со статическим методом generate(), возвращающим один из объектов Square, Circle, Box, Table. Реализацию выполнить с помощью паттерна фабрики по алгоритму Round Robin.

Создать класс Action со статическим методом, принимающим объект и выполняющим метод draw(), если тип объекта Drawable и move(), если тип объекта Movable.

В методе main создать 20 объектов с помощью генератора и выполнить статический метод в классе Action для каждого объекта

**Задача 3**

В данной задаче необходимо реализовать интерфейсы и абстрактные классы, которые позволят описать геометрические фигуры. Каждая фигура должна иметь 2 метода: double getLength(), вычисляющая длину окружности, кривой или периметр (в зависимости от фигуры) и double getArea(), вычисляющая площадь фигуры.

Все фигуры делятся на два типа – те, которые можно задать одним параметром и те, которые задаются двумя параметрами.

|  |  |  |
| --- | --- | --- |
| Фигура | Количество параметров для определения фигуры | Параметр(ы) определяет(ют) |
| Квадрат | 1 | Сторона квадрата |
| Круг | 1 | Радиус окружности |
| Равносторонний треугольник | 1 | Сторона треугольника |
| Прямоугольник | 2 | Длина и ширина прямоугольника |
| Эллипс | 2 | Длины полуосей |
| Прямоугольный треугольник | 2 | Длины катетов |

Реализовать абстрактный класс Figure.

Реализовать абстрактные классы OneParameterFigure, TwoParameterFigure.

Реализовать интерфейсы FigureLength, FigureArea, содержащие методы getLength() и getArea()

Реализовать классы Square, Circle, EquilateralTriangle, Rectangle, Ellipse, RightTriangle.

В классе RightTriangle (прямоугольный треугольник) реализовать метод double getHypotenuse(), который возвращает длину гипотенузы.

Все классы должны содержать конструктор без параметров, устанавливающий значения всех параметров случайным значением от 1 до 10.

Создать класс Generator, содержащий статический метод Figure generateFigure(), генерирующий одну из описанных фигур. Фигуры должны генерироваться по очереди (Round Robin).

В методе main создать список Figure, заполнить его 18 фигурами. Для каждого элемента данного списка вывести сообщения в формате:

Для фигур с одним параметром: <Класс фигуры> [<Значение параметра>]: Длина - <длина>, Площадь - <площадь>

Для фигур с двумя параметрами: <Класс фигуры> [<Значение параметра 1>, <Значение параметра 2>]: Длина - <длина>, Площадь - <площадь>.

После вывода данных о всех фигурах необходимо вывести данные о прямоугольных треугольниках в формате:

Гипотенуза треугольника [<Параметр1>, <Параметр2>] равна <Длина гипотенузы>.

Для всех чисел с дробной частью выводить только один знак после разделителя.

**Задача 4**

В данной задаче необходимо реализовать функционал ресторана. Необходимо реализовать меню, состоящее из нескольких блюд. У каждого блюда есть определенное время, которое оно готовится. В ресторане работает случайное число поваров (от 5 до 10), каждый из которых берет на себя работу по приготовлению блюда.

Реализовать абстрактный класс Food, обозначающий блюдо, с полем int cookingTime, определяющим время приготовления данного блюда.

Создать абстрактный класс Distributor, реализовать в нем интерфейс CookChooser, описывающей метод Cooker chooseCooker(List<Cooker> cookers). Добавить метод addFood(Food food, List<Cooker> cookers). Метод addFood должен выбирать повара и добавлять ему в список заказанное блюдо. Создать реализации данного абстрактного класса RoundRobinDistributor, BusyTimeDistributor. Данные классы должны реализовывать метод chooseCooker следующим образом: RoundRobinDistributor распределяет заказы между поварами по очереди, BusyTimeDistributor на основе загруженности каждого повара (заказ распределяется на повара, имеющего наименьшее время выполнения всех текущих заказов).

Создать класс Cooker (повар) с полем List<Food>, определяющим список блюд в очереди на приготовление данным поваром. Также класс должен предоставлять методы получения суммарного времени приготовления блюд, суммарного количества приготовления блюд, добавления блюда в список и очистки списка блюд для приготовления.

Реализовать классы на основе класса Food согласно таблице:

|  |  |
| --- | --- |
| Класс | cookingTime |
| Meat | 40 |
| Fish | 40 |
| Rice | 20 |
| Buckwheat | 20 |
| Coffee | 5 |
| Tea | 5 |

В методе main сгенерировать заказ на 100 блюд (выбирать случайно с равным шансом, использовать паттерн фабричный метод). Сравнить, сколько времени потребуется для приготовления данных блюд при использовании систем распределения RoundRobinDistributor и BusyTimeDistributor