ЗМІСТ

Відмітити та обґрунтувати недоліки проєктування 2
Визначити відповідність реалізації програми.....2
Зазначити недоліки реалізації.....2
Перевірити валідацію даних. 3
SOLID 5
Відмітити складність перебудови попередньої програми 6

Відмітити та обґрунтувати недоліки проєктування

1. **Відсутність ієрархії класів**:
   Класи *Road і ParallelCrossRoad* повинні бути похідними від загального базового класу, що містить загальну функціональність для всіх типів доріг. Це спростило б структуру коду та дозволило б зменшити дублювання коду при подальшому розширенні коду.
2. **Недостатнє використання інтерфейсів:**
   Використання інтерфейсів може полегшити розширення та заміну окремих компонентів системи. Наприклад, використання інтерфейсу для *TrafficLight* або створення *TrafficLightBase* дозволило б замінити різні типи світлофорів без зміни коду.

Визначити відповідність реалізації програми

В цілому, код виконує необхідні функції які були вказані у ТЗ.

Зазначити недоліки реалізації

1. **Симулятор не є гнучким**.
   Кількість смуг, світлофорів і доріг є харкодом, відповідно він не може бути використаний для більш обширного тестування чи при інших паттернах дороги.
2. **Не зовсім зрозуміло, чому і дорога, і смуга мають світлофори**.
   Можуть бути незручності або потенційне місце для багів, адже не зовсім зрозуміло, до котрого світлофора звертатись.
3. **Додавання до світлофора створює нову смугу.**
   На мою думку буде швидше і оптимальніше по пам’яті створити метод для додавання/заміни світлофора для певної смуги.
4. **Road.ToString() не оптимальний.**
   Не використовуються String.Join або StringBuilder, відповідно при більших масштабах буде багато сміття із string-ів які ми постійно створюємо.

5. **<u>Івенти створені, але ніяк не використовуються.</u>**
   Добре що є шаблон їх потенційного використання, але у конкретній реалізації вони ніяк не використовуються, а одже займають пам'ять і погіршують читабельність коду. Також при реалізації івентів для Line, на мою думку, логічніше використати *LineEntered & LineLeft*.
6. **<u>ParallelCrossRoad має неоднозначне ім'я і багато повторюваного коду.</u>**
   З приводу ім'я, дороги можуть бути не під 90 градусів або йти не строго по сторонам світу, при цьому керуватись тією самою логікою. Також не зовсім зрозуміло дорога паралельна чому? Краще просто перейменувати у *Crossroad*.
7. **<u>ParallelCrossRoad.AddTrafficLightInLine()</u>**
   Не зовсім зрозуміло, що саме виконує функція і навіщо створювати масив.
8. **<u>CrossRoadManger.Work() викликає Start()</u>**
   На мою думку зручніше перейменувати *Work() y Start()* для більшої читабельності, зручності і стандартизованого виду.

Перевірити валідацію даних

1. *public Road(string name, TrafficLight trafficLight, int countLines)*
   Перевірки відсутні. Потрібно додати
   - if (string.IsNullOrEmpty(name))
   - if (trafficLight == null)
   - if (countLines <= 0)
2. *Road.AssignTrafficLight(int currentLine, TrafficLight trafficLight)* – усі необхідні перевірки реалізовані.
3. *public Line(string name, TrafficLight trafficLight = null, double width = 2.5d)*
   Є перевірка на trafficLight, але відсутня для Name і Width
4. *Line.Start() і Line.Stop()* – твій задум передбачає, що у смуги світлофор може бути відсутнім (null-ем). Ти перевіряєш це при кожному виклику цих функцій, через що ти пишеш багато однакового коду, тому краще перевіряти це всередині функції, аніж зовні.

5. *public ParallelCrossRoad(string name, Road northSouth, Road southNorth, Road eastWest, Road westEast)* – публічний, тому тут потрібні перевірки на null-ові значення доріг і валідність імені.
6. *public void AddTrafficLightInLine(string roadName, int line, TrafficLight trafficLight)* – в цілому ок, тільки додай перевірку валідності імені.
7. *public CrossRoadManager() і AddScheme()* – немає перевірки на нуль, при тому що List може прийняти і зберегти їх
8. *public TrafficLight() I public TrafficLightWithTurn()* – немає перевірки на інтервал <= 0

SOLID

Принцип	Приклад гарної реалізація	Приклад порушення
SRP	*TrafficLight*	*Road* – і об’єкт дороги, і ініціалізує смуги, і контролює світлофори
OCP	Добре розділені і інкапсульовані поля і методи класів	
LSP	Немає дочірніх класів, які явно порушують принцип Лісков	
ISP	В цілому код слідує принципу ISP	Публічні методи, що ніде не використовуються (наприклад *CrossRoadManager.Stop()*) потенційно порушують принцип ISP
DIP		Хардкод Симулятора і *ParallelCrossRoad* створює нерозривний зв’язок між цими классами, при тому що симулятор повинен приймати будь-які дороги

Відмітити складність перебудови попередньої програми

1. *ParallelCrossRoad*
   Повністю змінена структура класу
2. *IschemeCrossRoads*
   Без змін
3. *RoadManager*
   Клас був повністю видалений і замінений на більш гнучкий клас CrossRoadManager, який може контролювати декілька доріг, замість однієї специфічної
4. *TrafficLight*
   Клас був повністю перероблений для можливості наслідування від нього, адже у початковій версії не було передбачено відповідної абстракції

Як підсумок, дуже мала частка оригінального коду була збережена. Функціонал майже повністю був переписаний з нуля, так само як і більша абстракція була створена вже в останній версії.