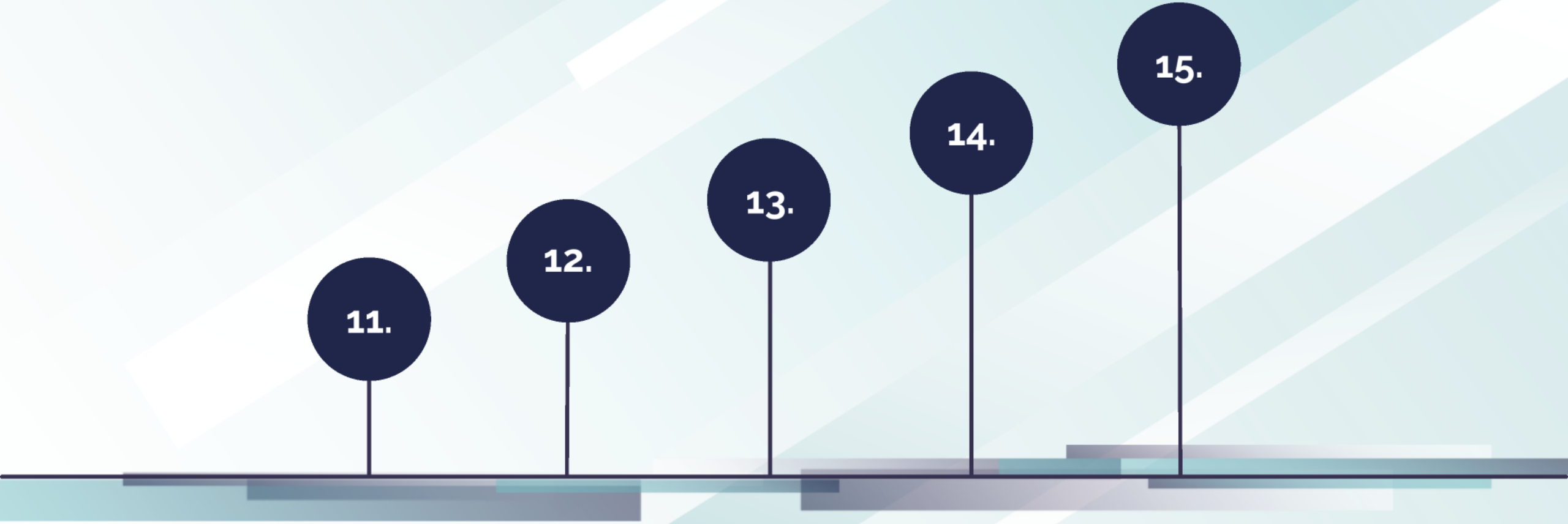
클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄



11. 기능 편애 (Feature Envy)

- **한 모듈이나 클래스가 다른 모듈이나 클래스의 기능이나 데이터에 지나치게 의존하는 상황**
- 문제점
  - 다른 클래스의 세부 사항에 지나치게 의존해서 코드 이해와 유지보수가 어려워짐.
- 해결법
  - 관련 데이터가 많은 클래스로 함수를 이동시킴.

Example 1

Example 2

예제 코드 1

```
// 안 좋은 코드 예시 (1)
0 references
public class Employee {
    1 reference
    private Department _department;

    0 references
    public int CalculateBonus() {
        return _department.GetBonusBase() * 2;
    }
}

1 reference
public class Department {
    1 reference
    public int GetBonusBase() {
        // 복잡한 계산...
        return 100;
    }
}
```

안 좋은 예시

```
// 좋은 코드 예시 (1)
2 references
public class Employee {
    2 references
    private int _bonusBase;

    0 references
    public Employee(int bonusBase) {
        _bonusBase = bonusBase;
    }

    0 references
    public int CalculateBonus() {
        return _bonusBase * 2;
    }
}

2 references
public class Department {
    // 기능은 Department 내부에 유지
}
```

좋은 예시

11. 기능 편애 (Feature Envy)

- **한 모듈이나 클래스가 다른 모듈이나 클래스의 기능이나 데이터에 지나치게 의존하는 상황**
- 문제점
  - 다른 클래스의 세부 사항에 지나치게 의존해서 코드 이해와 유지보수가 어려워짐.
- 해결법
  - 관련 데이터가 많은 클래스로 함수를 이동시킴.

Example 1

Example 2

예제 코드 2

```
// 안 좋은 코드 예시 (2)
0 references
public class Order {
    2 references
    private Customer _customer;

    0 references
    public decimal CalculateOrderDiscount() {
        if (_customer.HasLoyaltyDiscount()) {
            return _customer.GetLoyaltyDiscount();
        }
        return 0;
    }
}

1 reference
public class Customer {
    1 reference
    public bool HasLoyaltyDiscount() {
        // 충성도 할인 여부를 판단하는 로직
        return true;
    }

    1 reference
    public decimal GetLoyaltyDiscount() {
        // 충성도 할인 금액 계산 로직
        return 0.1m; // 10% 할인
    }
}
```

안 좋은 예시

```
// 좋은 코드 예시 (2) - 전략 패턴 사용
3 references
public interface IDiscountStrategy {
    2 references
    decimal CalculateDiscount();
}

1 reference
public class LoyaltyDiscountStrategy : IDiscountStrategy {
    3 references
    private Customer _customer;

    0 references
    public LoyaltyDiscountStrategy(Customer customer) {
        _customer = customer;
    }

    2 references
    public decimal CalculateDiscount() {
        // 충성도 할인 금액 계산 로직
        if (_customer.HasLoyaltyDiscount()) {
            return _customer.GetLoyaltyDiscount();
        }
        return 0;
    }
}

2 references
public class Order {
    2 references
    private IDiscountStrategy _discountStrategy;

    0 references
    public Order(IDiscountStrategy discountStrategy) {
        _discountStrategy = discountStrategy;
    }

    0 references
    public decimal CalculateOrderDiscount() {
        return _discountStrategy.CalculateDiscount();
    }
}
```

좋은 예시

11. 기능 편애 (Feature Envy)

- **한 모듈이나 클래스가 다른 모듈이나 클래스의 기능이나 데이터에 지나치게 의존하는 상황**
- 문제점
  - 다른 클래스의 세부 사항에 지나치게 의존해서 코드 이해와 유지보수가 어려워짐.
- 해결법
  - 관련 데이터가 많은 클래스로 함수를 이동시킴.

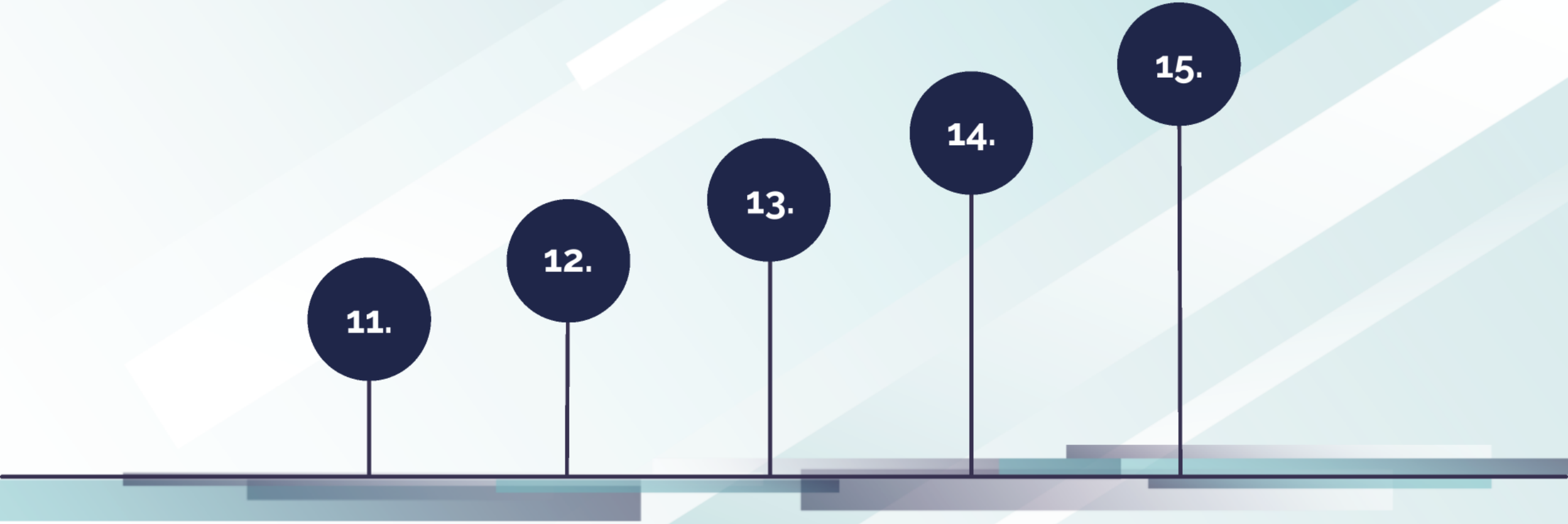
Example 1

Example 2

클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄



12. 데이터 뭉치 (Data Clumps)

- **서로 밀접하게 관련된 데이터들끼리 자주 함께 나타나는 현상** (예: 주소, 도시, 우편번호가 함께 전달된다거나)
- **문제점**
  - 코드의 중복을 증가시킴
- **해결법**
  - 서로 관련된 필드는 새로운 클래스로 분리함.
  - 함수에 전달되는 매개변수가 항상 같이 나타난다면, 객체로 묶어서 전달함.

Example

예제 코드 1

```
// 안 좋은 코드 예시 (1)
1 reference
public class Order {
    0 references
    public void ProcessOrder(string customerName, string address, string city, string postalCode) {
        // 주문 처리 로직...
    }
}
```

안 좋은 예시

```
// 좋은 코드 예시 (1)
1 reference
public class CustomerAddress {
    0 references
    public string Name { get; set; }
    0 references
    public string Address { get; set; }
    0 references
    public string City { get; set; }
    0 references
    public string PostalCode { get; set; }
}

1 reference
public class Order {
    0 references
    public void ProcessOrder(CustomerAddress address) {
        // 주문 처리 로직...
    }
}
```

좋은 예시

12. 데이터 뭉치 (Data Clumps)

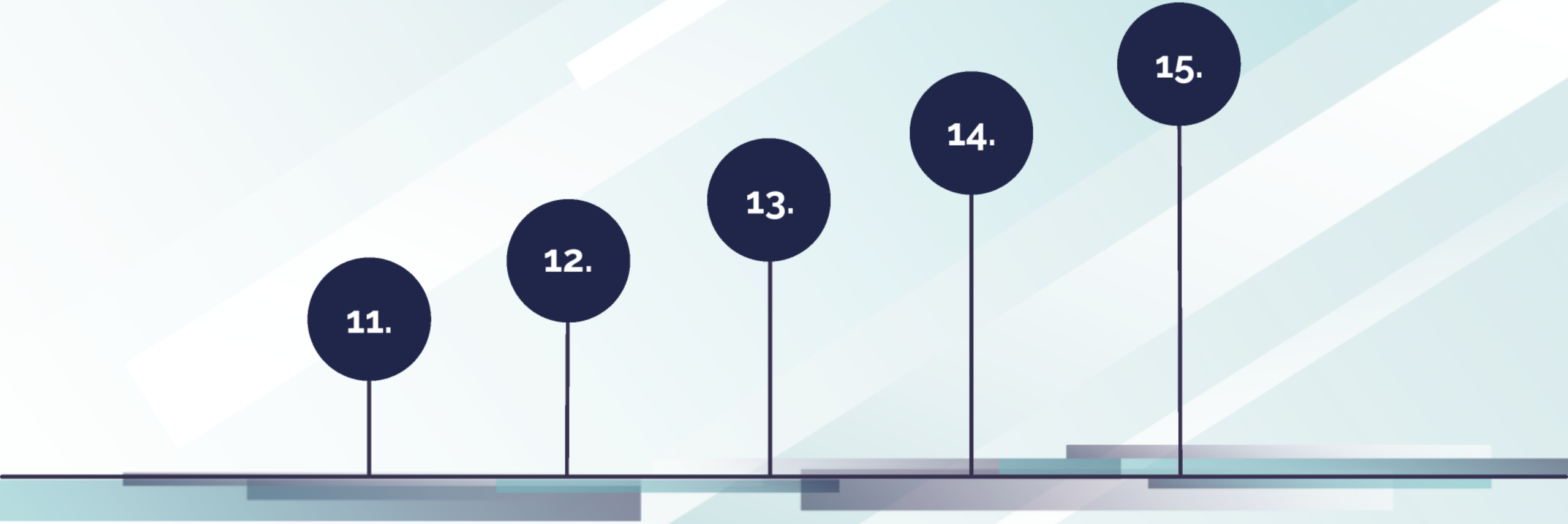
- **서로 밀접하게 관련된 데이터들끼리 자주 함께 나타나는 현상** (예: 주소, 도시, 우편번호가 함께 전달된다거나)
- **문제점**
  - 코드의 중복을 증가시킴
- **해결법**
  - 서로 관련된 필드는 새로운 클래스로 분리함.
  - 함수에 전달되는 매개변수가 항상 같이 나타난다면, 객체로 묶어서 전달함.

Example

클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄



13. 반복문 (Loops)

- 반복문은 매우 유용하지만, 때로는 코드가 복잡해지고 가독성이 떨어짐.
- 요즘의 트렌드는 '파이프라인'이라는 개념을 제시하는 추세임. (LINQ 기능)
- 반복문은 명령형 프로그래밍. 어떻게?를 명시
- 파이프라인은 선언형 프로그래밍. 무엇을?을 명시함.
- 적절하게 반복문과 파이프라인을 번갈아 사용

Example 1

Example 2

예제 코드 1

```
// 안 좋은 코드 예시 (1)
List<int> numbers = new List<int> {1, 2, 3, 4, 5};
List<int> evenNumbers = new List<int>();
foreach (var number in numbers) {
    if (number % 2 == 0) {
        evenNumbers.Add(number);
    }
}
```

안 좋은 예시

```
// 좋은 코드 예시 (1)
List<int> numbers = new List<int> {1, 2, 3, 4, 5};
var evenNumbers = numbers.Where(n => n % 2 == 0).ToList();
```

좋은 예시

13. 반복문 (Loops)

- 반복문은 매우 유용하지만, 때로는 코드가 복잡해지고 가독성이 떨어짐.
- 요즘의 트렌드는 '파이프라인'이라는 개념을 제시하는 추세임. (LINQ 기능)
- 반복문은 명령형 프로그래밍. 어떻게?를 명시
- 파이프라인은 선언형 프로그래밍. 무엇을?을 명시함.
- 적절하게 반복문과 파이프라인을 번갈아 사용

Example 1

Example 2

예제 코드 2

```
// 안 좋은 코드 예시 (2)
List<string> names = new List<string> {"Alice", "Bob", "Charlie"};
List<string> upperCaseNames = new List<string>();
foreach (var name in names) {
    upperCaseNames.Add(name.ToUpper());
}
```

안 좋은 예시

```
// 좋은 코드 예시 (2)
List<string> names = new List<string> {"Alice", "Bob", "Charlie"};
var upperCaseNames = names.Select(name => name.ToUpper()).ToList();
```

좋은 예시

13. 반복문 (Loops)

- 반복문은 매우 유용하지만, 때로는 코드가 복잡해지고 가독성이 떨어짐.
- 요즘의 트렌드는 '파이프라인'이라는 개념을 제시하는 추세임. (LINQ 기능)
- 반복문은 명령형 프로그래밍. 어떻게?를 명시
- 파이프라인은 선언형 프로그래밍. 무엇을?을 명시함.
- 적절하게 반복문과 파이프라인을 번갈아 사용

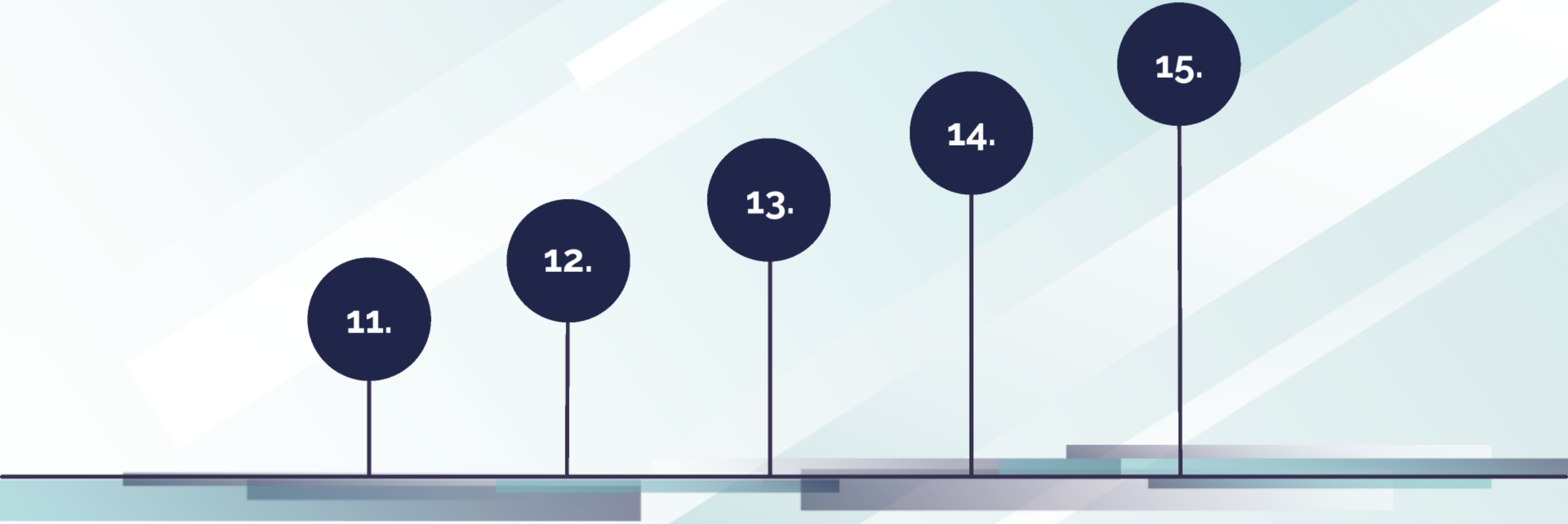
Example 1

Example 2

클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄



14. 임시 필드 (Temporary Field)

- 특정 상황에서만 사용되는 필드가 존재하는 경우를 말함
- 클래스의 다른 부분에서는 사용되지 않거나 값이 할당되지 않은 그런 값들.
- 해결법
  - 임시 필드가 있는 경우는 별도의 클래스나 메소드로 만들기
  - 임시 필드 관련된 함수는 다른 곳으로 옮기기

Example

예제 코드 1

```
// 안 좋은 코드 예시 (1)
1 reference
public class Customer {
    0 references
    public string Name { get; set; }
    1 reference
    public decimal CreditLimit; // 임시 필드
    0 references
    public DateTime? LastPurchaseDate; // 임시 필드

    0 references
    public void UpdateCreditLimit(decimal newLimit) {
        CreditLimit = newLimit;
    }
}
```

안 좋은 예시

```
// 좋은 코드 예시 (1)
1 reference
public class Customer {
    0 references
    public string Name { get; set; }
    0 references
    public CreditInfo CreditInformation { get; set; }
}

1 reference
public class CreditInfo {
    1 reference
    public decimal CreditLimit { get; set; }
    0 references
    public DateTime? LastPurchaseDate { get; set; }

    0 references
    public void UpdateCreditLimit(decimal newLimit) {
        CreditLimit = newLimit;
    }
}
```

좋은 예시

14. 임시 필드 (Temporary Field)

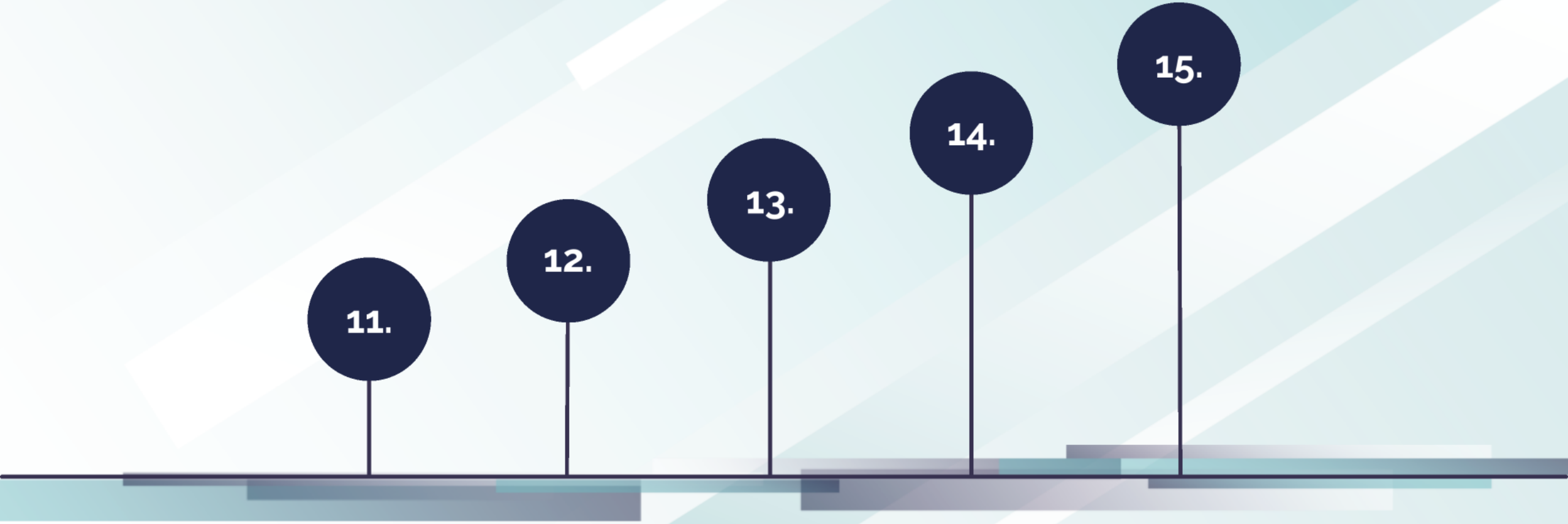
- 특정 상황에서만 사용되는 필드가 존재하는 경우를 말함
- 클래스의 다른 부분에서는 사용되지 않거나 값이 할당되지 않은 그런 값들.
- 해결법
  - 임시 필드가 있는 경우는 별도의 클래스나 메소드로 만들기
  - 임시 필드 관련된 함수는 다른 곳으로 옮기기

Example

클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄



15. 상속 포기하기 (Refused Bequest)

- **자식 클래스가 부모 클래스로부터 받은 메서드나 데이터를 일부만 사용하는 경우**
- **해결법**
  - 부모 클래스에서 자식 클래스로만 필요한 메소드와 필드를 옮긴다
  - 상속 대신 delegation를 사용한다. 즉, 자식 클래스가 부모 클래스의 인스턴스를 내부에 가지고 있게 하는 방식임.

Example 1

Example 2

예제 코드 1

```
// 안 좋은 코드 예시 (1)
6 references
public class Vehicle {
    2 references
    public void StartEngine() { /* 엔진 시작 로직 */ }
    2 references
    public void StopEngine() { /* 엔진 정지 로직 */ }
    0 references
    public void Fly() { /* 비행 로직 - 모든 차량이 날 수는 없음 */ }
}

1 reference
public class Car : Vehicle {
    // Car 클래스는 Fly 메소드를 필요로 하지 않음
}
```

안 좋은 예시

```
public class Vehicle {
    2 references
    public void StartEngine() { /* 엔진 시작 로직 */ }
    2 references
    public void StopEngine() { /* 엔진 정지 로직 */ }
}

0 references
public class FlyingVehicle {
    2 references
    private Vehicle _vehicle = new Vehicle();
    0 references
    public void Fly() { /* 비행 로직 */ }

    0 references
    public void StartEngine() {
        _vehicle.StartEngine();
    }

    0 references
    public void StopEngine() {
        _vehicle.StopEngine();
    }
}

1 reference
public class Car {
    2 references
    private Vehicle _vehicle = new Vehicle();

    0 references
    public void StartEngine() {
        _vehicle.StartEngine();
    }

    0 references
    public void StopEngine() {
        _vehicle.StopEngine();
    }
}
```

좋은 예시

15. 상속 포기하기 (Refused Bequest)

- **자식 클래스가 부모 클래스로부터 받은 메서드나 데이터를 일부만 사용하는 경우**
- **해결법**
  - 부모 클래스에서 자식 클래스로만 필요한 메소드와 필드를 옮긴다
  - 상속 대신 delegation를 사용한다. 즉, 자식 클래스가 부모 클래스의 인스턴스를 내부에 가지고 있게 하는 방식임.

Example 1

Example 2

예제 코드 2

```
// 안 좋은 코드 예시 (2)
4 references
public class Employee {
    1 reference
    public void Work() { /* 일반적인 작업 */ }
    0 references
    public void TakeVacation() { /* 휴가 사용 */ }
}

1 reference
public class Intern : Employee {
    // 인턴은 휴가를 사용하지 않음
}
```

안 좋은 예시

```
// 좋은 코드 예시 (2)
4 references
public class Employee {
    1 reference
    public void Work() { /* 일반적인 작업 */ }
    0 references
    public void TakeVacation() { /* 휴가 사용 */ }
}

1 reference
public class Intern {
    1 reference
    private Employee _employee = new Employee();

    0 references
    public void Work() {
        _employee.Work();
    }

    // TakeVacation 메소드는 포함하지 않음
}
```

좋은 예시

15. 상속 포기하기 (Refused Bequest)

- **자식 클래스가 부모 클래스로부터 받은 메서드나 데이터를 일부만 사용하는 경우**
- **해결법**
  - 부모 클래스에서 자식 클래스로만 필요한 메소드와 필드를 옮긴다
  - 상속 대신 delegation를 사용한다. 즉, 자식 클래스가 부모 클래스의 인스턴스를 내부에 가지고 있게 하는 방식임.

Example 1

Example 2

클린코드 시리즈

김하연 튜터
챌린지반 특강 6강

안 좋은 코드란? 3탄