**Orientação a objetos**

* Coesão:
  + Uma classe coesa faz bem uma única coisa
  + Classes coesas não devem ter várias responsabilidades
* Encapsulamento:
  + Getters e setters não são formas eficientes de aplicar encapsulamento
  + É interessante fornecer acesso apenas ao que é necessário em nossas classes
  + O encapsulamento torna o uso das nossas classes mais fáceis e intuitivo
* Acoplamento:
  + Acoplamento é a dependência entre classes
  + Acoplamento nem sempre é ruim, e que é impossível criar um sistema sem nenhum acoplamento
  + Devemos controlar o nível de acoplamento na nossa aplicação

**Melhorando a coesão**

* Que classes/métodos/funções/módulos devem ter uma única responsabilidade bem definida;
* Que, segundo o Princípio de Responsabilidade Única (SRP), uma classe deve ter um e apenas um motivo para ser alterada;
* Como realizar uma refatoração no nosso sistema, para aplicar o SRP;
* Como extrair uma classe.

**Reduzindo o acoplamento**

* Que cada classe deve conhecer e ser responsável por suas próprias regras de negócio;
* Que o princípio Aberto/Fechado (OCP) diz que um sistema deve ser aberto para a extensão, mas fechado para a modificação
  + Isso significa que devemos poder criar novas funcionalidades e estender o sistema sem precisar modificar muitas classes já existentes
* Uma classe que tende a crescer "para sempre" é uma forte candidata a sofrer alguma espécie de refatoração.

**Herança indesejada**

* Que, embora a herança favoreça o reaproveitamento de código, ela pode trazer efeitos colaterais quando não utilizada da maneira correta;
* Que o Princípio de Substituição de Liskov (LSP) diz que devemos poder substituir classes base por suas classes derivadas em qualquer lugar, sem problema.

**Trabalhando com abstrações**

* Que é mais interessante e mais seguro para o nosso código depender de interfaces (classes abstratas, assinaturas de métodos e interfaces em si) do que das implementações de uma classe;
* Que as interfaces são menos propensas a sofrer mudanças enquanto implementações podem mudar a qualquer momento;
* Que o Princípio de Inversão de Dependência (DIP) diz que implementações devem depender de abstrações e abstrações não devem depender de implementações;
* Que as interfaces devem definir apenas os métodos que fazem sentido para seu contexto;
* Que o Princípio de Segregação de Interfaces (ISP) diz que uma classe não deve ser obrigada a implementar um método que ela não precisa;
* Os conceitos aprendidos neste treinamento formam o acrônimo SOLID
  + Single Responsibility Principle
  + Open Closed Principle
  + Liskov Substitution Principle
  + Interface Segregation Principle
  + Dependency Inversion Principle