1. d; 2) a; 3)b; 4) a; 5) c; 6) a; 7) c; 8) b; 9) d

10) Alguns conceitos da oop – programação orientada a objetos, são:  
Herança: quado uma classe “filha” herda atributos e metodos de outras classes “super class”. Em Java, através da palavra frase extends na definição da classe descendente, ela permite o reaproveitamento do que foi herdado.   
Polimorfismo: quando temos um ponto de entrada único porém a classe se comporta ou executa de forma diferente dependendo do parâmetro de entrada  
Abstração: quando escondemos os detalhes da implementação do usuário e provemos apenas a funcionalidade principal. Em Java criamos usamos a palavra reservada abstract em sua definição.  
Encapsulamento: quando restringimos o acesso aos atributos e metodos de uma classe de forma que podemos controlar o que será exposto para outras classes que queiram reutilizá-la. Em java podemos utilizar os modificadores private por exemplo.

11) Sim. MVC ou Model View Control agrega boas práticas no desenvolvimento de uma aplicação através do desacoplamento entre essas camadas. Ou seja, separamos a camada Model (camada de dados e regra de negócio), View (camada de apresentação das informações que interagem com o usuário), Controller (trata o fluxo das informações entre as demais camadas) de forma que seja de fácil manutenção e organização do código fonte.

12) Sim. Design patterns ou padrões de desenho apresentam soluções para os problemas comuns através das boas práticas de qualidade, reutilização de código, facilidade na manutenção do código e redução do TCO.   
Os autores conhecidos como Gang of Four publicaram o livro Design Patterns em 1994 e posteriormente temos o JavaEE Design Pattern. Através desses conceitos temos várias opções de soluções arquiteturais para os problemas mais comuns para os projetos, são eles:

**Creational** ou de Criação (para instanciar componentes),   
**Structural** ou Estrutural (para tratarmos estrutura de dados simples e complexas de forma bem organizada) e   
**Behavioral** ou de Comportamental (para prover baixo acoplamento e a interação entre os componentes)

13) Abstract Factory permite a criação de fábricas de outras fábricas (Factories) abstraindo a implementação ou a parte concreta desse componente que pode fabricar outros componentes que podemos instanciar a partir de um parâmetro de entrada.   
Builder também é um padrão de criação porém mais adequado para problemas mais complexos, com muitos parâmetros de entrada e que podem ser inconsistentes.

14) Façade é um padrão estrutural que permite provermos uma interface que encapsula toda a complexidade de interação entre diversos recursos externos, fontes de dados e relacionamento entre API´s diferentes

15) O Command é um padrão comportamental adequado para utilizarmos quando precisamos disparar uma ação ou melhor executar uma operação de forma que possamos encapsular toda complexidade do lado do cliente. O Command é específico para uma funcionalidade, pois conhece tudo que é necessário para executar e deve prover o método execute() na sua chamada.

16) Sim. O acrônimo SOLID representa cinco princípios da maturidade arquitetural, boas praticas e padrões de qualidade indispensáveis ao desenvolvimento de software. São eles:  
**Single Responsibility** (responsabilidade única, ou seja, reduzir o acoplamento entre responsabiliades diferentes),   
**Open/Closed** (Aberto para extensão e fechado para modificação, ou seja, deve permitir adicionar funcionalidades ao invés de alterar o que já está estável e com funcionamento seguro),   
**Liskov** (prover a substituição das subclasses através da super class),   
**Interface** (prover interfaces bem definidas, simples e específicas para cada necessidade), Dependency inversion (prover independencia entre os módulos)

17) Sim. Entendo que o DDD (Domain-Driven Design) é um desenvolvimento orientado a dominio aborda uma forma de trabalho onde o foco é o negócio então ao interagirmos com o usuário final e adquirimos o conhecimento sobre o que é e como funciona a regra.

18) Sim. Entendo que TDD (Test-Driven Design) é um desenvolvimento orientado a testes, ou seja, visando boas práticas e qualidade de software, é uma abordagem que devemos ter ao pensar nos testes unitários ou fim a fim e precisa fazer parte do ciclo de vida desde o inicio do desenvolvimento da aplicação.

19) Sim. Entendo que ORM (Object Relational Mapping) é o mapeamento de classes Java em entidades JPA para representar as tabelas da base de dados afim de permitir obtermos as funcionalidades do CRUD (Create, Read, Update and Delete) de forma ágil que quando dispensamos a implementação do SQL mais complexo e sofisticado.

20) Sim. Meu conhecimento superficial sobre SOA (Service-Oriented Architecture) é de uma arquitetura orientada ao serviço de forma independente do fabricante ou da tecnologia a ser utilizada. Nessa abordagem o foco é o serviço e utiliza protocolos de comunicação que são utilizados ao transportar mensagens usando metadados, prevalecendo sempre o baixo acoplamento.

21) Orientação a aspecto, conhecida por AOP (Aspect-oriented programming) tem como principal objetivo a separação dos modulos de um projeto de acordo com sua importância, ou seja, o que é primordial ou secundário. Por exemplo é quando separamos o parse das informações, a validação da regra de negócio, verificação dos parâmetros informados e demais necessidades do que é realmente é essêncial para a funcionalidade principal.

22) A DI (Dependency Injection) ou injeção de dependencia é um design pattern para tratarmos a IoC (Inversion Of Control) ou Inversão de Controle. Prefiro exemplificar considerando uma classe A depende da classe B e tratamos essa dependencia na prática utilizando um construtor, um método Set ou através de uma interface. Entendo que sempre que injetamos um objeto estamos considerando o baixo acoplamento entre essas dependencias.