Antropomorfismo – O que o objeto sabe e faz, fazer essa pergunta para determina quais são os atributos e métodos quando se criar uma classe. Ambos tipos são as responsabilidades da classe.

Quando uma classe precisa da outra para ser executada, fala que essa outra classe é a colaboradora e suas responsabilidades são colaborações (Exemplo uma Classe extrato precisa de uma classe registroDeOperacoes)

Cartão CRC – Classe, Responsabilidade e Colaboração é um cartão formado por estes três, cada classe tem apenas um cartão aonde coloca todas suas responsabilidades (atributos e métodos) e coloca também quais classes colaboradoras e quais colaborações ela precisa.

Programação Orientada a Objeto é semelhante a construção de um lego, o programa é dividido em vários pedaços e no final esses pedaços são ‘montados’ construindo o programa final. Assim facilita a manutenção futura do programa.

Para importar métodos e variáveis estáticas utiliza o comando:

Import static java.util.Math.\*

A modelagem CRC é composta por 6 passos mas antes de começar a desenvolver esses 6 passos, precisamos ter o passo ‘zero’ que seria a especificação do sistema (requisitos).

Passo 1: Identificar as classes da aplicação, fazendo uma lista com todos substantivos e nomes da especificação, todos esses substantivos escolhidos são considerados candidatos a classes na aplicação. A lista será refinada, podendo existir substantivos com nomes diferentes mas que representam o mesmo substantivo e também substantivos que podem ser considerados atributos de classes. Um exemplo seria biblioteca e nome da biblioteca, são dois candidatos a classes (foram inseridos na lista) porem o segundo candidato é um atributo do primeiro. Descrever cada classe que restar e verificar se existe algum relacionamento (herança) entre os substantivos restantes. Após isso devemos transformar todos substantivos em singular. Elimina os atributos (características de outros substantivos), elimina também os sinônimos (substantivos com nomes diferentes mas com o mesmo significado). Após esses passos os atores devem ser eliminados, cuidado, pois atores são usuários q irão utilizar o sistema mas que não precisam ser armazenados nada deles no sistema.

Passo 2: Uma breve descrição do que significa a classe e começa a criação do cartão CRC adicionando o nome da classe e a descrição.

Passo 3: Identificamos responsabilidades obvias, exemplo em uma classe cliente, algumas possíveis responsabilidades obvias serão cadastro de cliente, consultar, alterar. Mesmo que não conseguimos descobrir todas responsabilidades obvias nesse passo, não tem problema, futuramente essas responsabilidades irão aparecer e serão adicionadas no cartão CRC. (lembrando que responsabilidades são atributos e métodos).

Passo 4: Identificar os verbos do tipo faz da especificação e listar todos eles, esse passo começamos a tentar identificar possíveis responsabilidades de métodos, verbos do tipo sabe são responsabilidades de atributos.

Passo 5: Pegando a lista de verbos do passo 4, vamos identificar a que classe pertence o verbo selecionado, basicamente vamos decidir qual classe que receberá o método selecionado.

Passo 6: Uma vez definido todas as classes e todos métodos dessas classe, nesse passo vamos determina qual a lógica dos métodos, como será a execução desses métodos. Nesse passo pode aparecer novas responsabilidades, as de colaborações (responsabilidades de outras classes que a atual precisará para executar a sua responsabilidade). Por isso que é muito importante fazer a descrição da lógica, justamente para encontrar essas novas responsabilidades.

Testando Software.

Um dos pontos mais positivos de testes automatizado é na hora de modificações no sistema, após a modificação basta apenas executar todos os testes ara certificar que nenhum método foi afetado.

A cada nova funcionalidade adicionada no código deve ser criado o seu teste unitário e executado este teste e os testes das funcionalidades antigas, para certificar que as outras funcionalidades não foram afetadas por esta nova.

Existe funcionalidades que não conseguimos realizar os testes automatizado, nesse caso precisa ser feito manualmente, porem sempre devemos utilizar os automatizados quando possível.

Vamos falar sobre três tipos de testes:

1. Teste de Unidade: Testa uma única classe ou único método.
2. Teste de Integração: Testa a colaboração (acomplamento) de um grupo de classe.
3. Teste Funcional: Aonde é testado o software por um todo, no ponto de visto do usuário.

Esses três tipos de teste não existe uma diferença técnica na criação, apenas no escopo, o modo de criar os teste é exatamente o mesmo, única diferença é a quantidade de classes, métodos que se utiliza, essa quantidade que determina o tipo de teste que está sendo utilizado.

Class Junit Anotações:

Para determina que um método é de teste utiliza-se a anotação @Test em cima do método (cada método terá sua anotação)

As anotações @Before e @After são para determina métodos que serão chamado antes e depois dos métodos de testes, importante, para cada método de teste será chamado o método que está anotado com @Before antes da execução e após a execução é chamado o método que está anotado com @After. Se tiver 10 métodos de teste, o before e after será chamado 10 vezes cada. Para chamar apenas 1 vez o before e o after se utiliza a anotação @BeforeClass e @AfterClass. Os métodos do BeforeClass e AfterClass precisam ser static para serem executado.

Diagrama UML

A vantagem do diagrama é em visualizar o projeto todo através de um mapa, se precisarmos entender algum fluxo do código, pode ser muito cansativo e complicado descobrir isto apenas pela codificação. Descobrir fluxo por diagramas é muito mais fácil e simples.

Dependência entre Classes

Se um objeto A não pode existir sem o objeto B, então existe uma dependência do objeto A em relação ao B. Exemplo, se a classe A utiliza qual método ou atributo da classe B, isso significa que a classe A depende da classe B, além desse exemplo, se uma classe utiliza uma outra classe como atributos e parâmetros também são consideradas dependência.

Classe cliente é a classe que irá utilizar a outra classe, podemos associar a classe cliente como a classe que não irá existir sem a outra classe. Classe servidora é a classe que é utilizada na classe cliente. Um exemplo seria em herança, a superclasse é uma classe servidora e as herdeiras são classes cliente, pois elas utilizam métodos da superclasse. Uma classe pode ser servidora dela mesmo, basta instanciar um outro objeto da mesma classe.

Encapsulamento

O ponto a destacar do encapsulamento é tomar cuidado para não passar uma referência e acabar quebrando ele, exemplo se temos uma lista de produtos em uma classe e passar essa lista, quem recebeu essa lista consegue modificar os dados, nesse caso não houve encapsulamento, nesse caso é interessante passar só o dado das posições ou fazer uma cópia da lista e passar a cópia. Cuidado também em referência de outros objetos também, se conseguirem modificar os valores do objeto, então o encapsulamento foi quebrado.

Polimorfismo

Quando uma classe é atribuída uma referência de sua superclasse ou interface, o interessante que por mais que ela seja atribuída em outro tipo de referência, a implementação que é usada é o que foi escrito no seus métodos e um detalhe apenas os métodos que a superclasse e a interface declaram que poderão ser utilizados mas a implementação será do objeto criado.

Law of Demeter conceito usado para diminuir as dependências das classes. Um método de um objeto só pode invocar métodos do próprio objeto, de parâmetros que ele recebeu, qualquer objeto que ele instancia e métodos de objetos que foram iniciados na classe, porem só se usa métodos DIRETOS, não pode usar métodos de objetos instanciados nos objetos que foram instanciados na classe atual.

Quando precisar utilizar algum objeto que não se enquadra nesse perfil, deve ser utilizado o principio de redirecionamento, aonde existe objetos q faz a logica desejada e repasse apenas os resultados, uma semelhança aos casos controler, models e dao.

Classe anônima é quando temos uma instancia de um objeto porem não temos uma referência direta a esse objeto.

Exceções

A hierarquia de exceções no java começa pela Throwable embaixo dela estão as classes Error e Exception, a sub classe de error é problemas externo do java, erros de hardware, sistema operacional, provavelmente não conseguimos tratar esses errors. Na subclasse Excepetion temos uma outra subclasse a RuntimeExcepetion todas classes que herdam da Runtime, não são obrigatórias trata-las como ArithmeticExcepetion, ClassCastExcepetion, NullPointExcepetion, IndexOutOfBoundsExcepetion entre outras. As outras subclasses de exceções que herdam da Excepetion como SQLExcepetion, IOExcepetion, PrintExcepetion, DataFormatExcepetion são excepetion que somos obrigados a tratar utilizando try/catch ou lançando para quem chamou o método.

StackTrace, é o caminho do erro, começando pela classe que deu erro até a última primeira classe que começou o ciclo até o error.

Nos blocos try/catch podemos colocar quantos catch necessários porem se acontecer algum erro apenas um catch vai ser executado, e será executado o primeiro catch que satisfaz o erro, podendo ser o próprio erro ou uma superclasse do erro. Se tivermos SQLExcepetion e dpois Excepetion, o catch com SQLExcepetion será executado, se invertemos Excepetion e depois SQLExcepetion, mesmo que o erro seja de SQL, como a Excepetion é superclasse e está primeiro, o bloco dela que será executado.

Temos também o finally que sempre será executado independente se deu erro ou não, mesmo que o método retorna algo, o finally sempre será executado.