Instituto de Computação uff

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrão Strategy

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

Um exemplo de problema

Considere o problema de criar um sistema para simulação de patos em um jogo

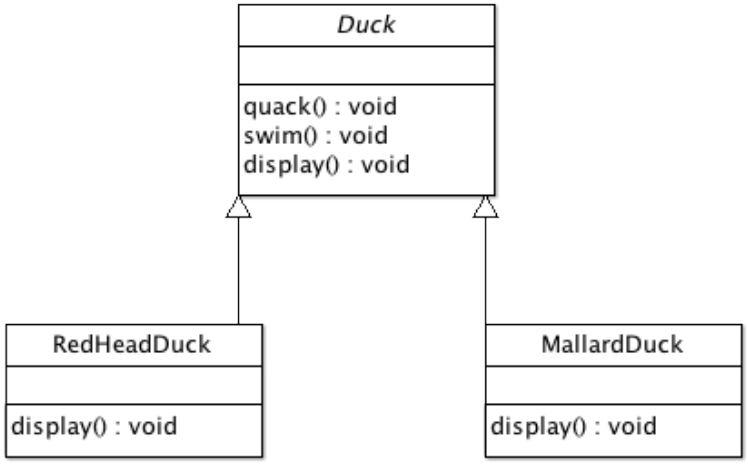
Os patos exibem os seguintes comportamentos:

- Grasnar (quack)
- Nadar (swim)
- Exibir (display)

O jogo deverá contemplar diferentes tipos de patos: Bravo, Cabeça-vermelha, Borracha, etc...

Padrões de Projeto

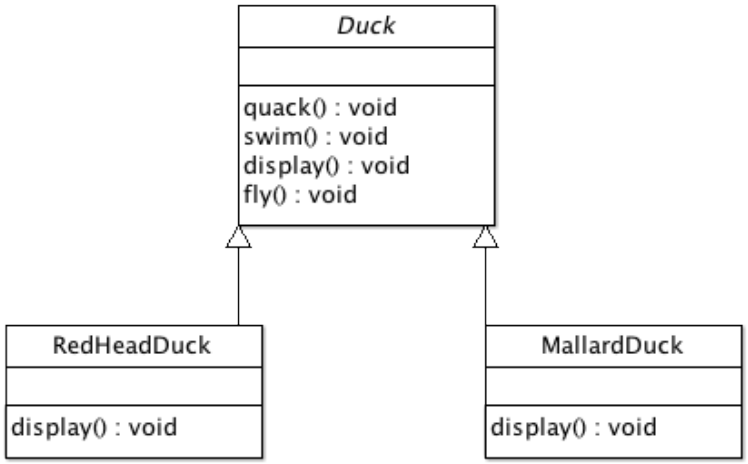
Uma primeira solução



Padrões de Projeto

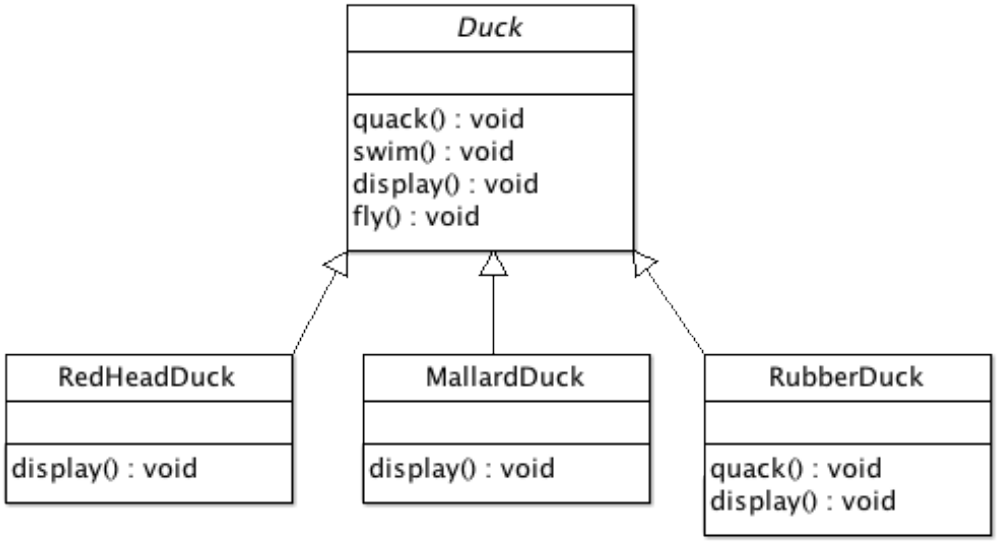
Uma primeira solução

Considere uma primeira necessidade de modificação: os patos tem que voar



Adicionar um método fly na superclasse parece uma boa solução...

Mas de repente surgem novos tipos de patos que não podem voar...



Herança parece não ser a solução quando não se conhece o comportamento de todos os patos...

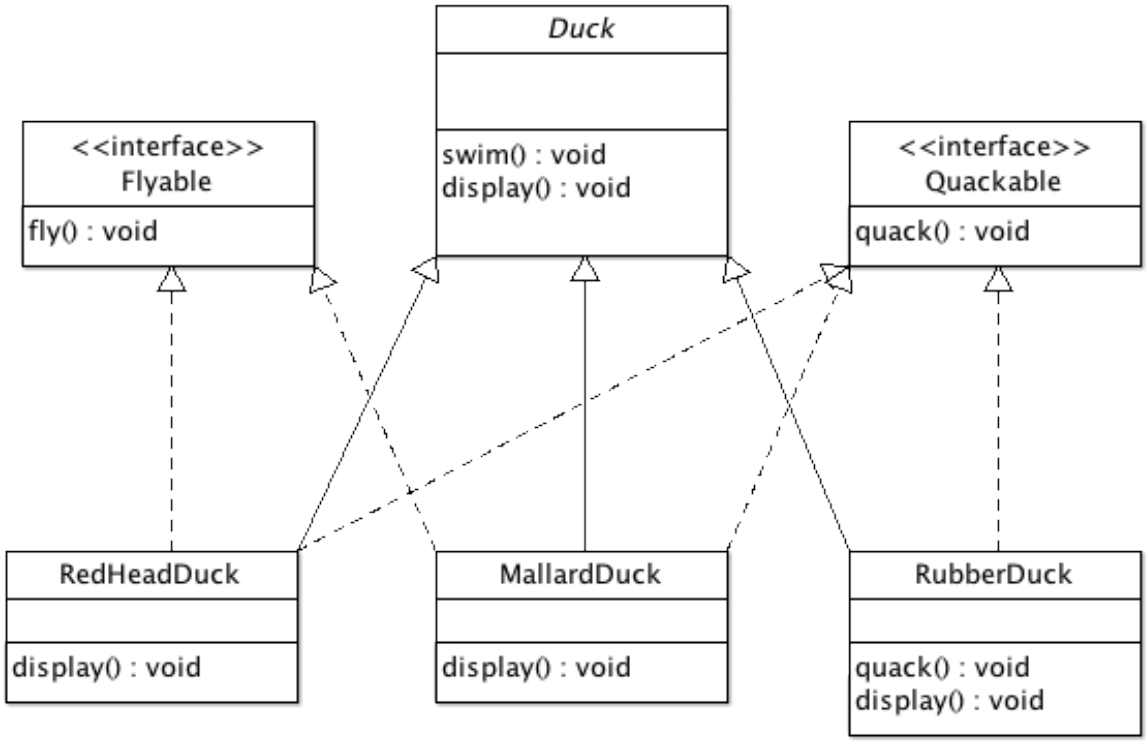
Problemas:

Não sabemos o comportamento de todos as classes(subtipos de patos) que podem fazer parte da simulação

A herança não permite alteração de comportamento em tempo de execução

As alterações podem afetar sem querer as subclasses de forma indesejada

Uso de Interfaces



Padrões de Projeto

Analisando ambas as soluções

Herança não funciona porque o comportamento das subclasses é imprevisível e não é apropriado que todas as subclasses tenham esses comportamentos

O uso de interfaces resolve o problema anterior mas destrói o reuso de código já que não possuem implementação

Como resolver esse problema?

Usando princípios de projeto OO

Primeiro Princípio

> Identifique os aspectos que variam e separe-os do que permanece igual

Pegue as partes que variam e encapsule-as para depois poder alterar ou estender as partes que variam sem afetar as que não variam

No problema da simulação dos patos, o que varia são os comportamentos Voar (fly()) e Grasnar (quack())

Então vamos separar esses comportamentos de pato e criar novas classes que implementam os dois comportamentos (voar e grasnar)

Também desejamos manter o processo flexível

Queremos pode atribuir diferentes comportamentos em tempo de execução aos patos

Para isso usaremos o segundo princípio de projeto O.O.

Segundo Princípio

Programe para uma interface, não para um implementação

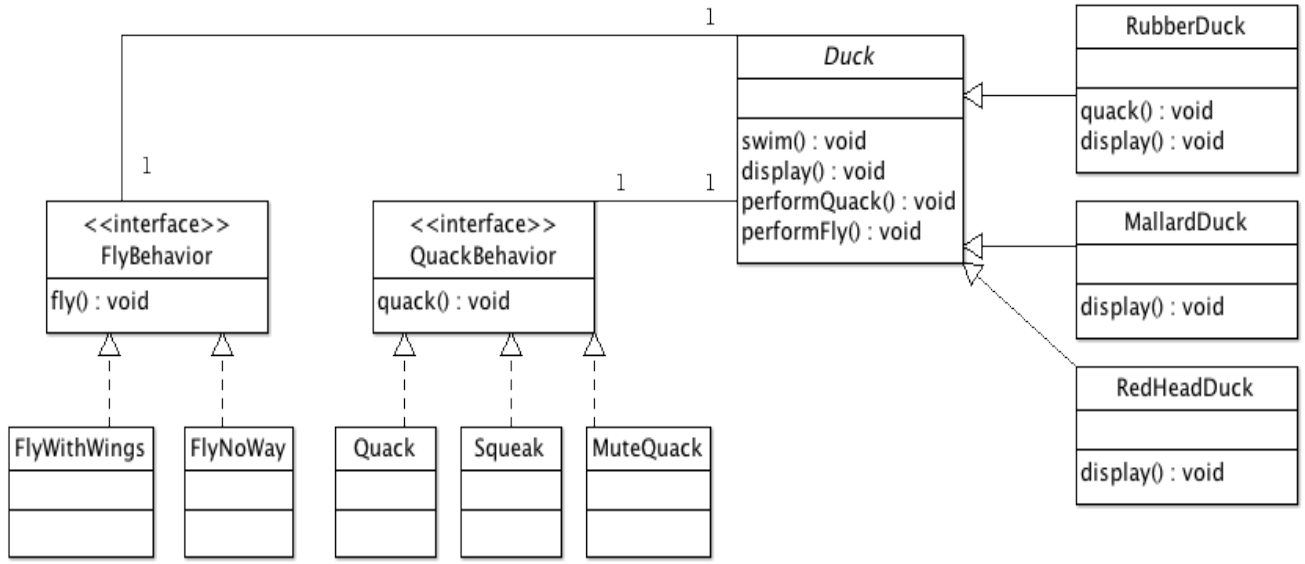
Programar para interface significa programar para um tipo

Iremos criar duas interfaces que representam os dois comportamentos que variam: FlyBehavior e QuackBehavior

Cada interface terá um conjunto de implementações representadas por diferentes classes

Padrões de Projeto

Solução via Princípios de O.O.



Agora os tipos de patos (MallardDuck, RedHeadDuck, RubberDuck) além de serem da classe Duck (relação é-um) também possuem atributos que descrevem seus comportamentos (relação tem-um)

Observe que estamos usando composição para adicionar comportamentos aos tipos de pato ao invés de fazê-los herdar tais comportamentos da classe mãe

Terceiro Princípio

Dar prioridade à composição

Uso de composição da forma vista acima permite encapsular uma família de algortimos

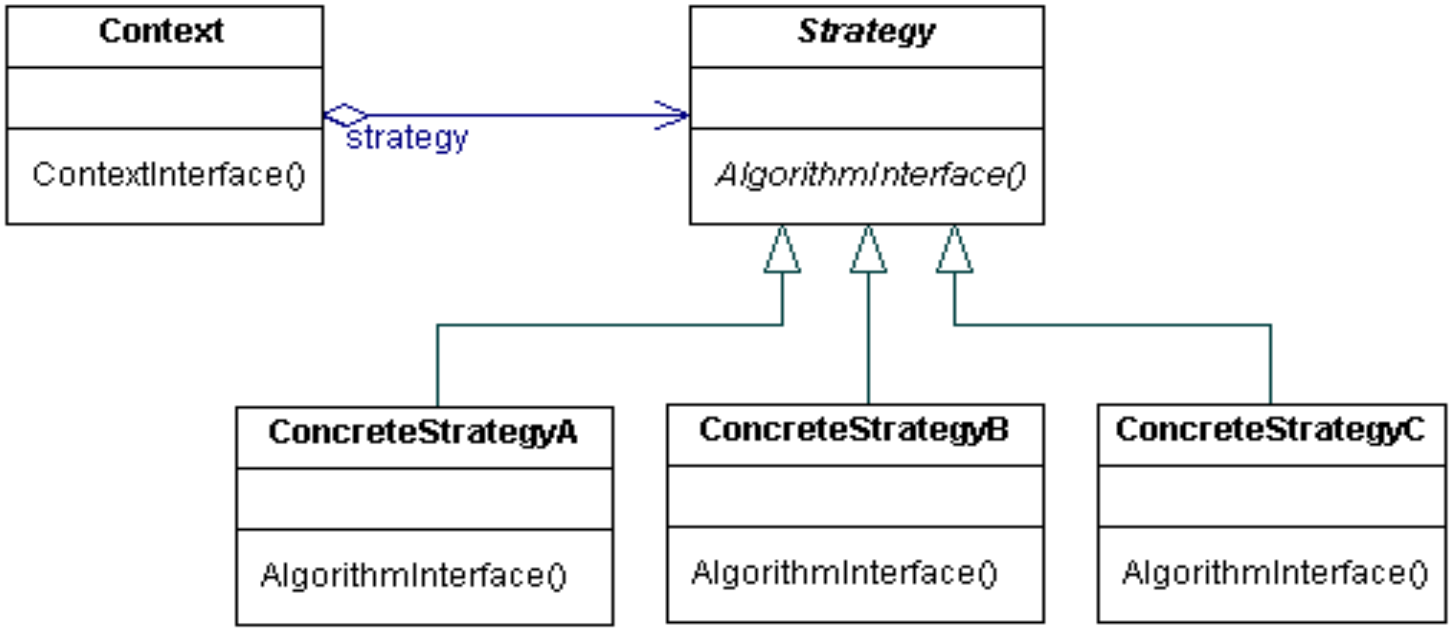
Também é possível alterar o comportamento em tempo de execução

Primeiro padrão: Strategy

Define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. A estratégia deixa o algoritmo variar independentemente dos clientes que o utilizam

Padrões de Projeto

O Padrão Strategy



Strategy

Define uma interface comum para todos os algoritmos suportados. Context usa esta interface para chamar o algoritmo definido por uma ConcreteStrategy.

ConcreteStrategy

Implementa o algoritmo usando a interface de Strategy.

Context

É configurado com um objeto ConcreteStrategy;

Mantém uma referência para um objeto Strategy;

Pode definir uma interface que permite a Strategy acessar seus dados.

Strategy

Define uma interface comum para todos os algoritmos suportados. Context usa esta interface para chamar o algoritmo definido por uma ConcreteStrategy.

ConcreteStrategy

Implementa o algoritmo usando a interface de Strategy.

Context

É configurado com um objeto ConcreteStrategy;

Mantém uma referência para um objeto Strategy;

Pode definir uma interface que permite a Strategy acessar seus dados.

Padrões de Projeto

Exercício: Implemente uma classe que descreve uma simulação de uma guerra entre dois países. Uma guerra envolve duas ações, declarar guerra e encerrar a guerra.

Uma vez declarada a guerra um país pode tomar diferentes ações de inicio e fim da guerra que dependem da característica do inimigo conforme a tabela apresentada no slide seguinte

Padrões de Projeto

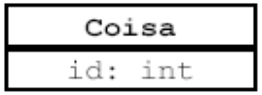
Inimigo	Estratégia	Iniciar	Concluir
Nuclear	Diplomacia	Recuar tropas Propor cooperação econômica	Desarmar inimigo
Grande exercíto (>10000 homens)	Aliança com vizinho	Atacar pelo Norte Vizinho Atacar pelo Sul	Dividir benefícios Divider custo de reconstrução
Frágil	Atacar sozinho	Plantar evidências falsas Lançar bombas Derrubar governo	Estabelecer governo Amigo

Obs.: Este exemplo tem o único objetivo de proporcionar uma atividade de programação em sala de aula. O autor dos slides em nenhum momento incentiva ou compactua com atividades bélicas.

- O método Arrays.sort (java.util) pode ser considerado um exemplo de Strategy. Ele recebe como parâmetro um objeto do tipo Comparator que implementa um método compare(a, b) e utiliza-o para definir as regras de ordenação.

```
public class MedeCoisas implements Comparator {
  public int compare(Object o1, Object o2) {
    Coisa c1 = (Coisa) o1;
    Coisa c2 = (Coisa) o2;
    if (c1.getID() > c2.getID()) return 1;
    if (c1.getID() < c2.getID()) return -1;
    if (c1.getID() == c2.getID()) return 0;
  }
}

...
Coisa coisas[] = new Coisa[10];
coisas[0] = new Coisa("A");
coisas[1] = new Coisa("B");
...
Arrays.sort(coisas, new MedeCoisas());
...
```

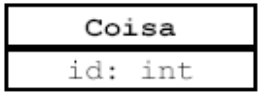


Método retorna 1, 0 ou -1 para ordenar Coisas pelo ID

- O método Arrays.sort (java.util) pode ser considerado um exemplo de Strategy. Ele recebe como parâmetro um objeto do tipo Comparator que implementa um método compare(a, b) e utiliza-o para definir as regras de ordenação.

```
public class MedeCoisas implements Comparator {
  public int compare(Object o1, Object o2) {
    Coisa c1 = (Coisa) o1;
    Coisa c2 = (Coisa) o2;
    if (c1.getID() > c2.getID()) return 1;
    if (c1.getID() < c2.getID()) return -1;
    if (c1.getID() == c2.getID()) return 0;
  }
}

...
Coisa coisas[] = new Coisa[10];
coisas[0] = new Coisa("A");
coisas[1] = new Coisa("B");
...
Arrays.sort(coisas, new MedeCoisas());
...
```



Método retorna 1, 0 ou -1 para ordenar Coisas pelo ID

Padrões de Projeto

Solução do exercício

```
public class Guerra {
   Estrategia acao;
   public void definirEstrategia() {
     if (inimigo.exercito() > 10000) {
        acao = new AliancaVizinho();
     } else if (inimigo.isNuclear()) {
        acao = new Diplomacia();
     } else if (inimigo.hasNoChance()) {
        acao = new AtacarSozinho();
     }
   }
   public void declararGuerra() {
      acao.atacar();
   }
   public void encerrarGuerra() {
      acao.concluir();
   }
}
```

```
public interface Estrategia {
   public void atacar();
   public void concluir();
}
```

```
public class AtacarSozinho
        implements Estrategia {
   public void atacar() {
      plantarEvidenciasFalsas();
      soltarBombas();
      derrubarGoverno();
   }
   public void concluir() {
      estabelecerGovernoAmigo();
   }
}
```

```
public class AliancaVizinho
        implements Estrategia {
   public void atacar() {
      vizinhoPeloNorte();
      atacarPeloSul();
      ...
   }
   public void concluir() {
      dividirBeneficios(...);
      dividirReconstrução(...);
   }
}
```

```
public class Diplomacia
        implements Estrategia {
   public void atacar() {
      recuarTropas();
      proporCooperacaoEconomica();
      ...
   }
   public void concluir() {
      desarmarInimigo();
   }
}
```

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books

- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman