Abstract Factory & Prototype

REGIS. Jardel <jardelbrandon@gmail.com>
FILHO. João <joao.santos@academico.ifpb.edu.br>

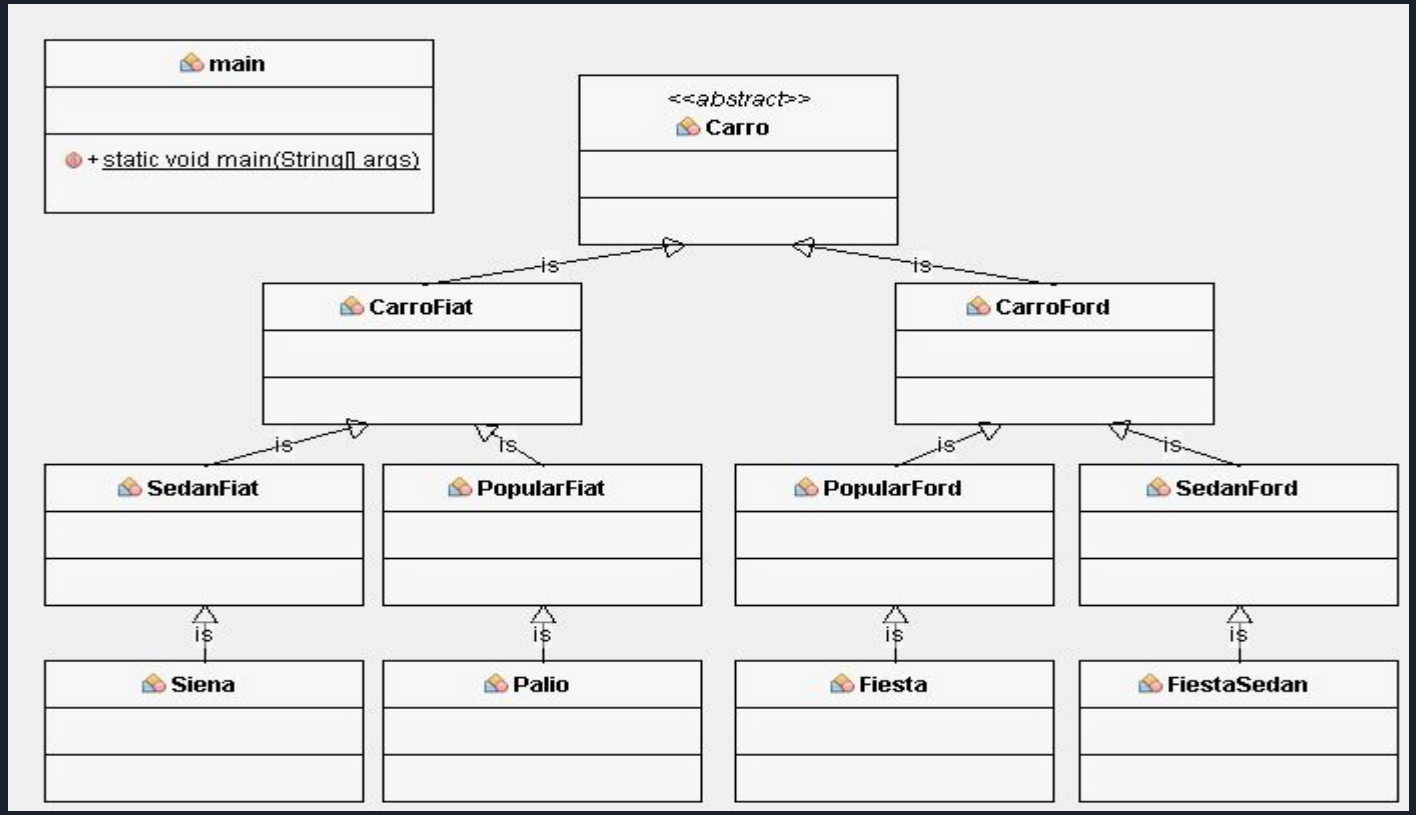
Introdução

“Abstract Factory - Fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.”[GoF]

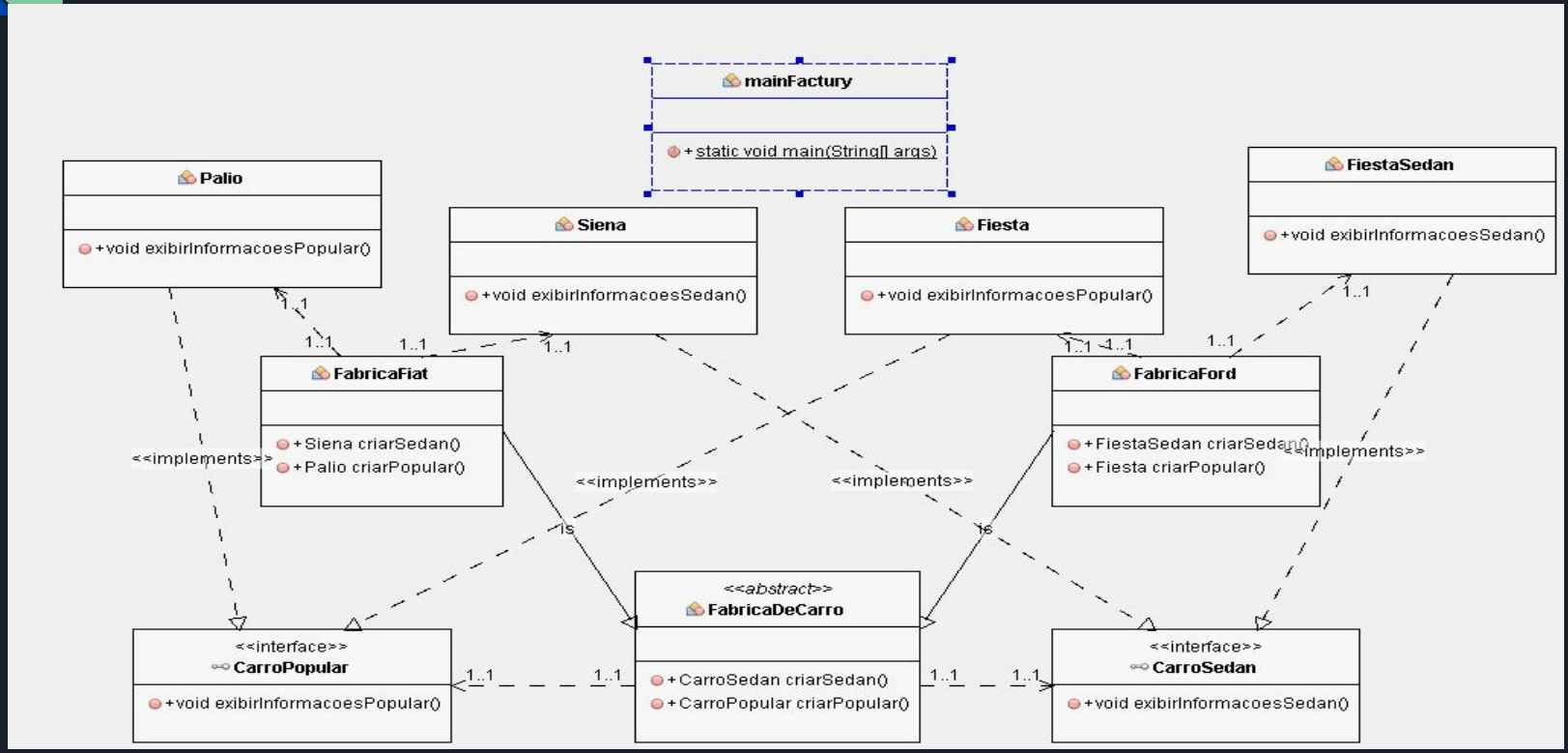
Problema

Queremos que, dado um conjunto de carros, seja possível instanciar carros com mesma marca, mas famílias distintas.

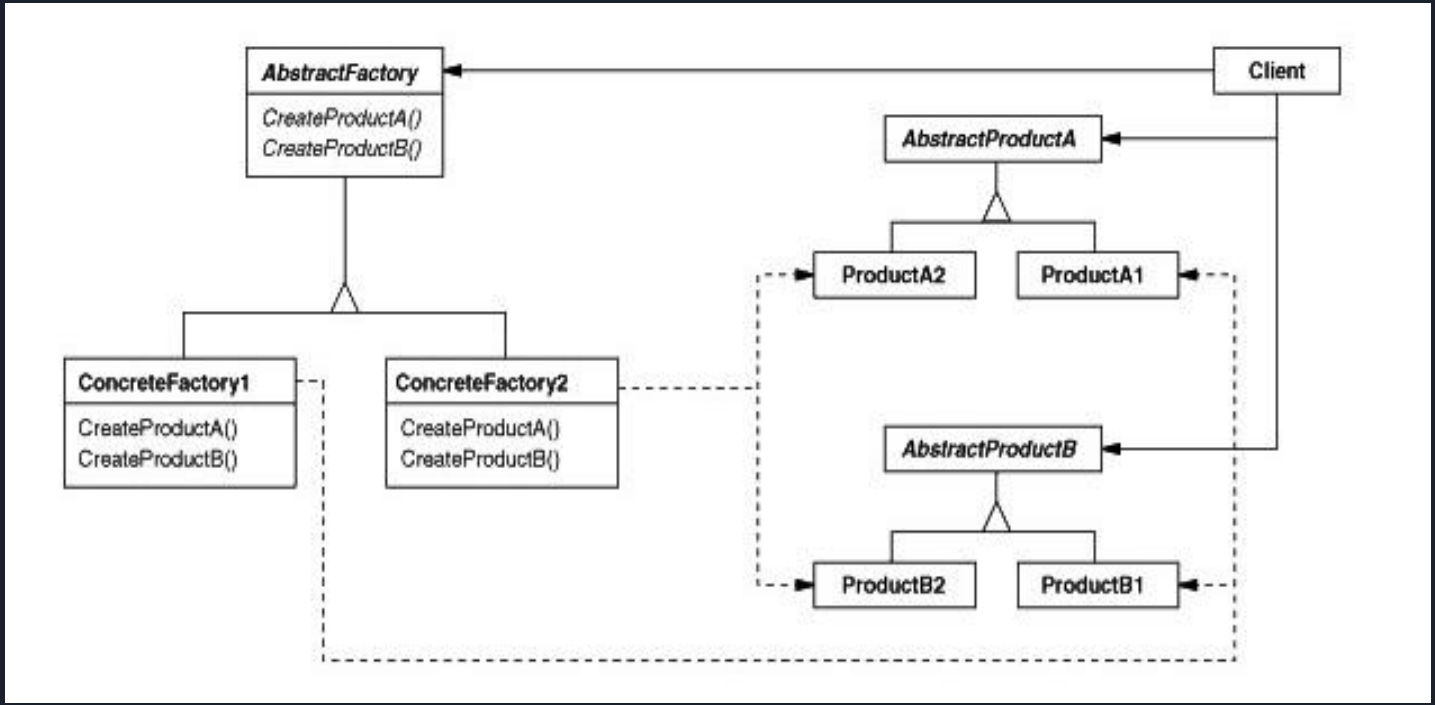
Solução Sem Padrão



Solução para o problema



Estrutura


AbstractFactory
CreateProductA()
CreateProductB()
Client
AbstractProductA
ProductA2
ProductA1
ConcreteFactory1
CreateProductA()
CreateProductB()
ConcreteFactory2
CreateProductA()
CreateProductB()
AbstractProductB
ProductB2
ProductB1

Participantes

- Fábrica Abstrata: Define a interface de criação do produto abstrato, bem como fatora o código comum as fábricas;

- Fábrica Concreta: Implementa as operações para criação de produtos concretos;

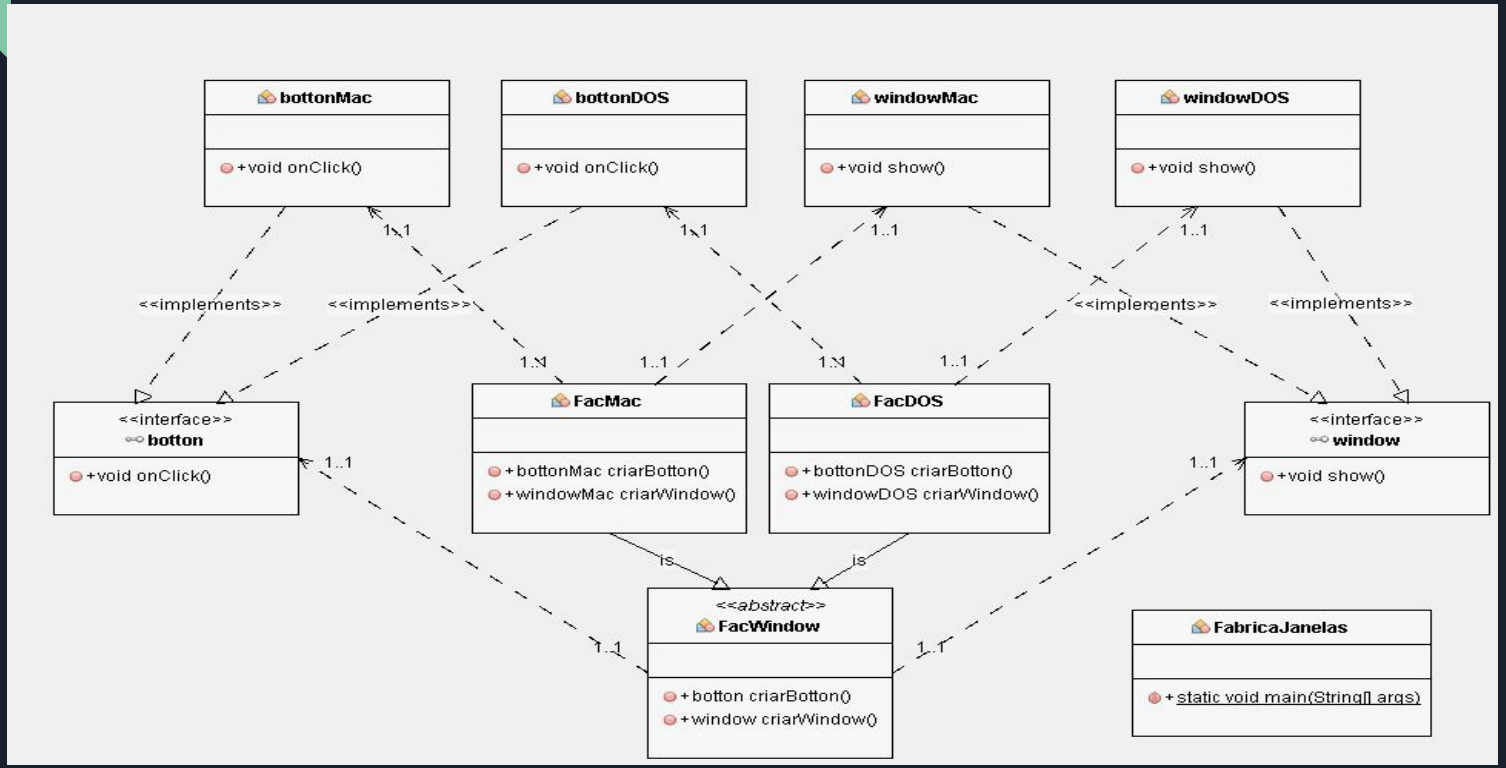
- Produto Abstrato: Define uma interface de objetos de um tipo, bem como fatora o código comum aos produtos concretos;

- Produto Concreto: Define um objeto produto concreto a ser criado pela fábrica concreta, além de implementar a interface ProdutoAbstrato.

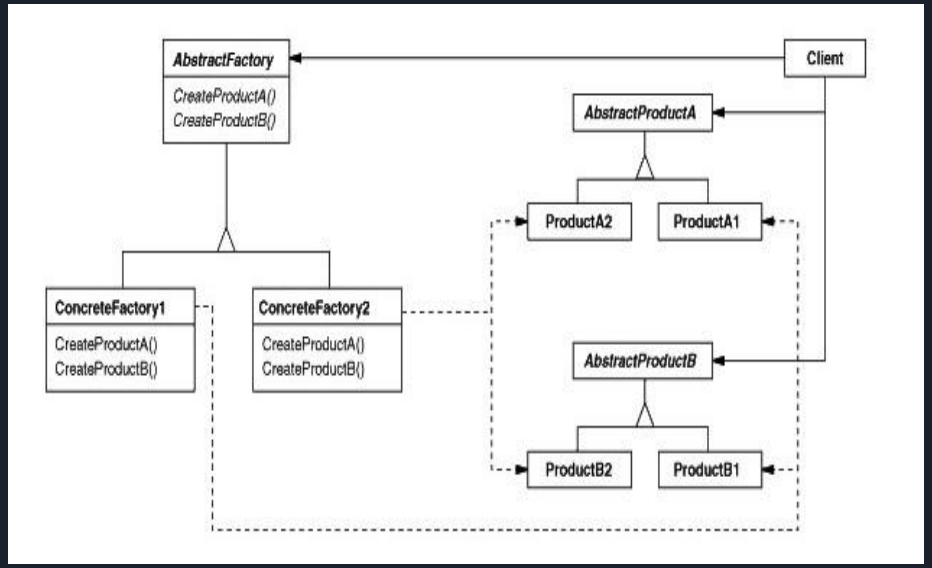
Quando usar?

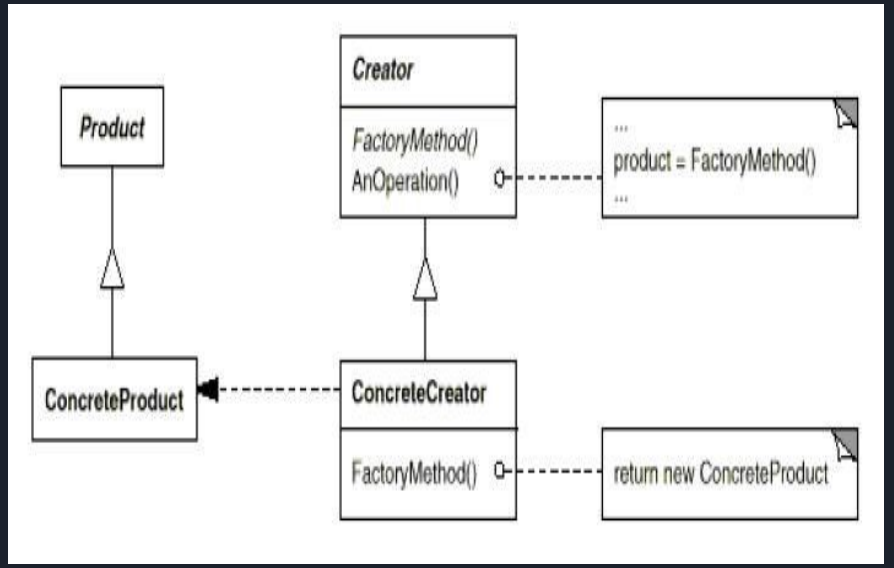
1. Quando um sistema deve ser independente de como seus produtos são criados, compostos e representados;
2. Quando um sistema deve ser configurado com uma entre várias famílias de produtos;
3. Quando uma família de produtos relacionados foi projetada para uso conjunto e você deve implementar essa restrição;
4. Quando você quer fornecer uma biblioteca de classes e quer revelar sua interface e não sua implementação.

Exemplo Janelas de Sistemas



Diferença em relação ao Factory Method





Vantagens e Desvantagens

- O padrão isola classes concretas. A factory encapsula a responsabilidade e o processo de criação de objetos, isolando clientes das classes de implementação.

- Produtos de uma determinada família devem funcionar conjuntamente e não misturados com os de outra família.

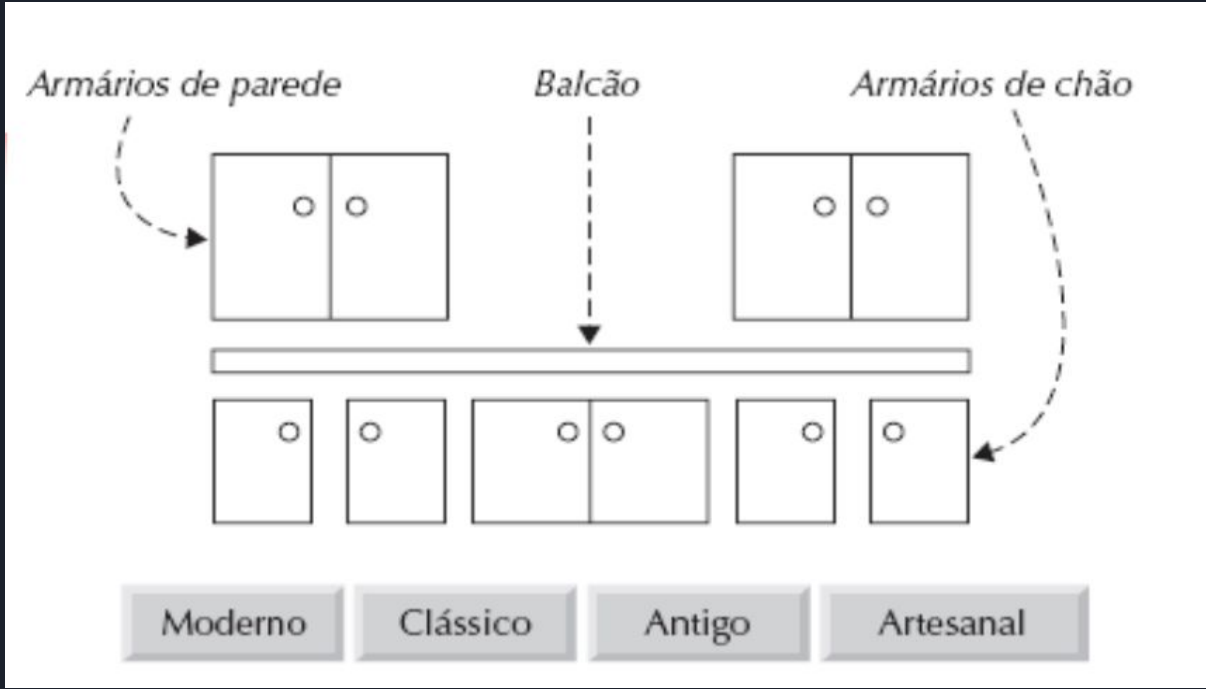
- Suportar novos tipos de produto exige estender a interface da fábrica, o que envolve mudar a classe AbstractFactory e todas as suas subclasses.

Relações com outros padrões

- Singleton;
- Factory Method;
- Facade;
- Composite;
- Prototype.

Questão

Imagine que temos uma aplicação para montagem de armários para cozinha. Nesta aplicação, é possível criar uma cozinha e nela organizar como será seu armário(posição e peças), as peças são armários de parede e de chão, de dois estilos: clássico e moderno. A atividade é propor a utilização do padrão Abstract Factory nesse contexto.


Armários de parede
Balcão
Armários de chão
Moderno
Clássico
Antigo
Artesanal

Prototype

"Especificar os tipos de objetos a serem criados usando uma instância como protótipo e criar novos objetos ao copiar este protótipo." [GoF]

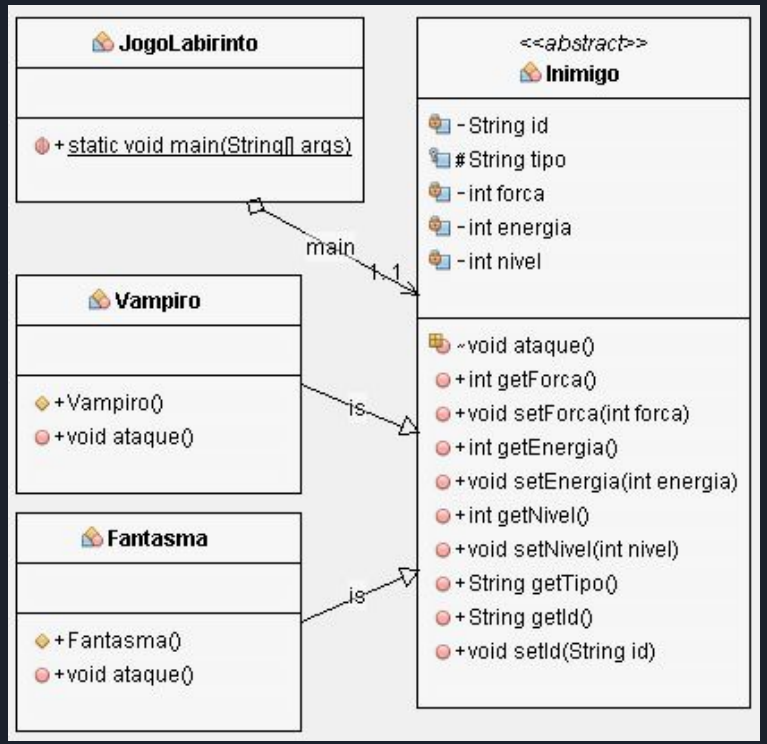
Problema

Exemplo: Um jogo de labirinto para computador

Queremos programar um jogo onde o jogador está em um labirinto repleto de monstros.

Inicialmente existem dois tipos de inimigos: Fantasmas e vampiros, imaginemos que uma classe abstrata Inimigo já foi criada, agora, cabe a nós implementarmos as classes concretas de cada inimigo e utilizá-las

Solução sem utilizar Design Patterns



Exemplo de código fonte da classe principal (main):

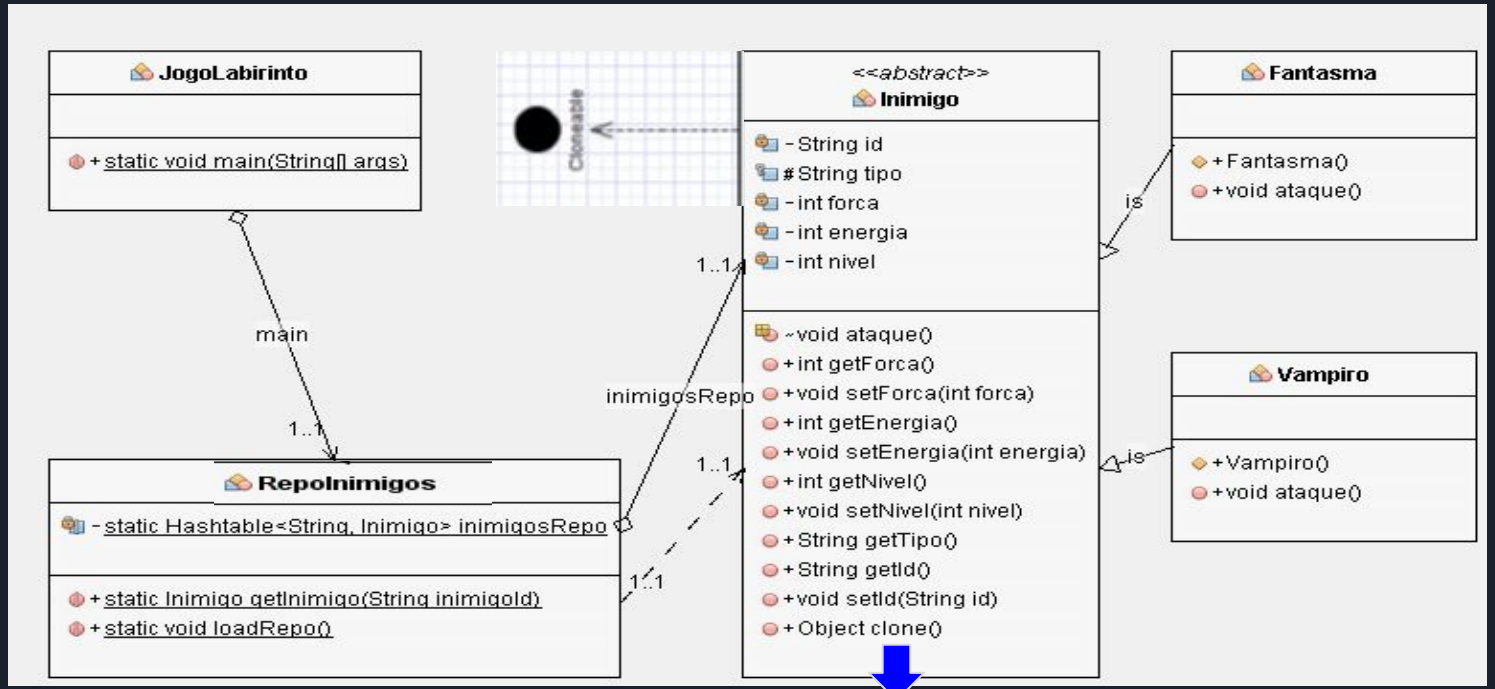
```
public static void main(String[] args) {
    Inimigo fantasma = new Fantasma();
    fantasma.setId("1");
    fantasma.setEnergia(10);
    fantasma.setForca(5);
    fantasma.setNivel(1);
    fantasma.ataque(); //Só aqui que poderá atacar

    Inimigo vampiro = new Vampiro();
    // …   Setup do novo vampiro      ...
    //Se mais na frente eu quiser mais inimigos,
    //Teremos mais repetições desse trecho
}
```

Consequências negativas

- Como pode ser visto a cada vez que se instanciar um inimigo tem que iniciá-lo corretamente antes de usá-lo se esta operação estiver relacionada outras operações mais custosas como acesso a banco de dados para preenchimento dos seus atributos já podemos contar com uma degradação de performance ali.
- E se o conjunto de atributos não varia para cada inimigo, por exemplo todo fantasma nível 1 tem 10 de energia e 20 de força, a repetição de código fica ainda mais redundante e desnecessária.
- Possibilidade de inserir erros (bugs) por não iniciar corretamente os inimigos.

Solução utilizando o padrão prototype




```
protected Object clone() throws CloneNotSupportedException {
      return super.clone();
}
```

Solução utilizando o padrão prototype

Exemplo de código fonte da classe RepoInimigos:

```
public class RepoInimigos {
   private static HashMap<String, Inimigo> inimigosRepo  = new HashMap<>();
   public static Inimigo getInimigo(String inimigoId)  {
       Inimigo protoInimigo = inimigosRepo.get(inimigoId);
       return (Inimigo) protoInimigo.clone();
   }
   public static void loadRepo()  {
       Fantasma fantasma = new Fantasma();
       fantasma.setId("1");
       fantasma.setEnergia(10);
       fantasma.setForca(5);
       fantasma.setNivel(1);
       inimigosRepo.put(fantasma.getId(),fantasma);
       Vampiro  vampiro = new Vampiro();
       //…    Setup do novo vampiro    ...
   }
}
```

Solução utilizando o padrão prototype

Exemplo de código fonte da classe principal (main):

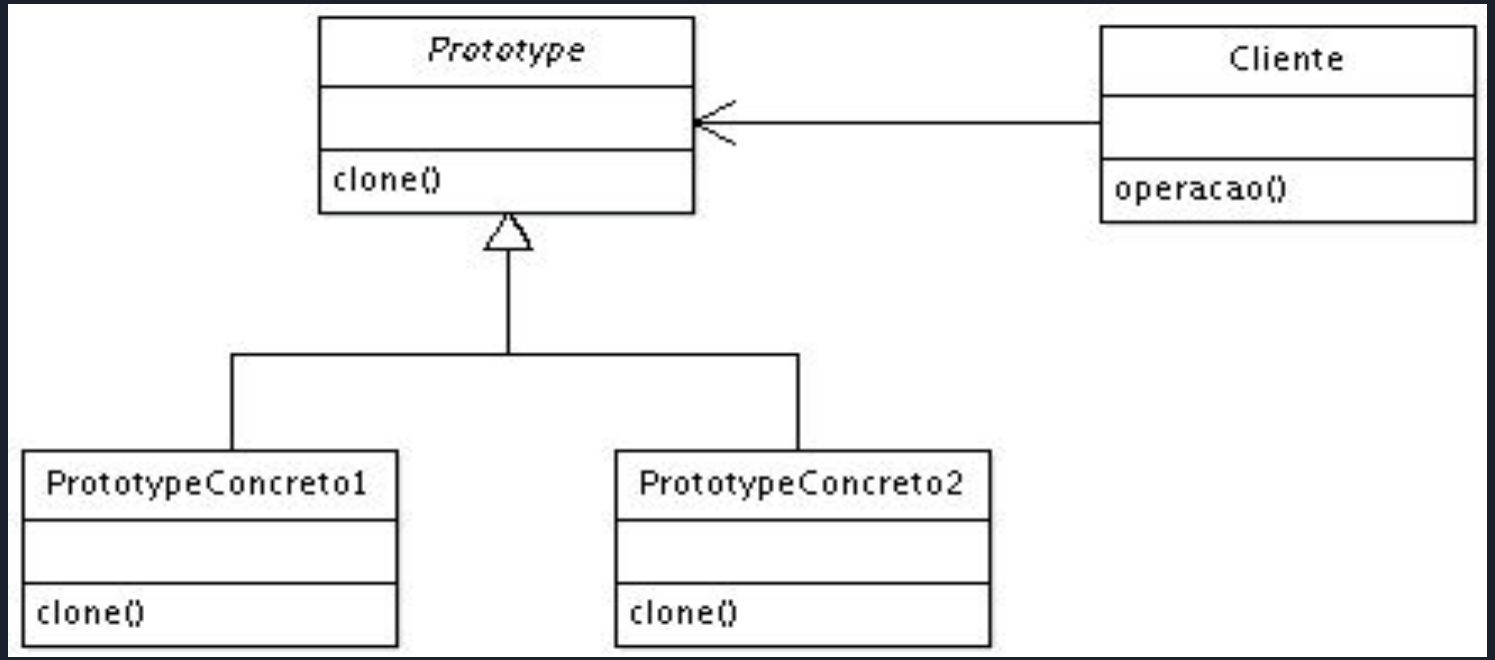
```
public class JogoLabirinto {
    public static void main(String[] args) {
        RepoInimigos.loadRepo();

        Inimigo fantasma_clone = (Inimigo) RepoInimigos.getInimigo("1");
        //Id = 1 = fantasma
        fantasma_clone.ataque(); //Fantasma já pode atacar pois o objeto veio completo!

        Inimigo vampiro_clone = (Inimigo) RepoInimigos.getInimigo("2");
        //Id = 2 = besta
        vampiro_clone.ataque(); // Vampiro já pode atacar pois o objeto veio completo!
    }
}
```

Conhecendo e aplicando o padrão prototype

Estrutura:



Participantes

- prototype — uma classe que declara uma interface para objetos capazes de clonar a si mesmo.

- prototype concreto — implementação de um prototype;

- cliente — cria um novo objeto através de um prototype que é capaz de clonar a si mesmo.

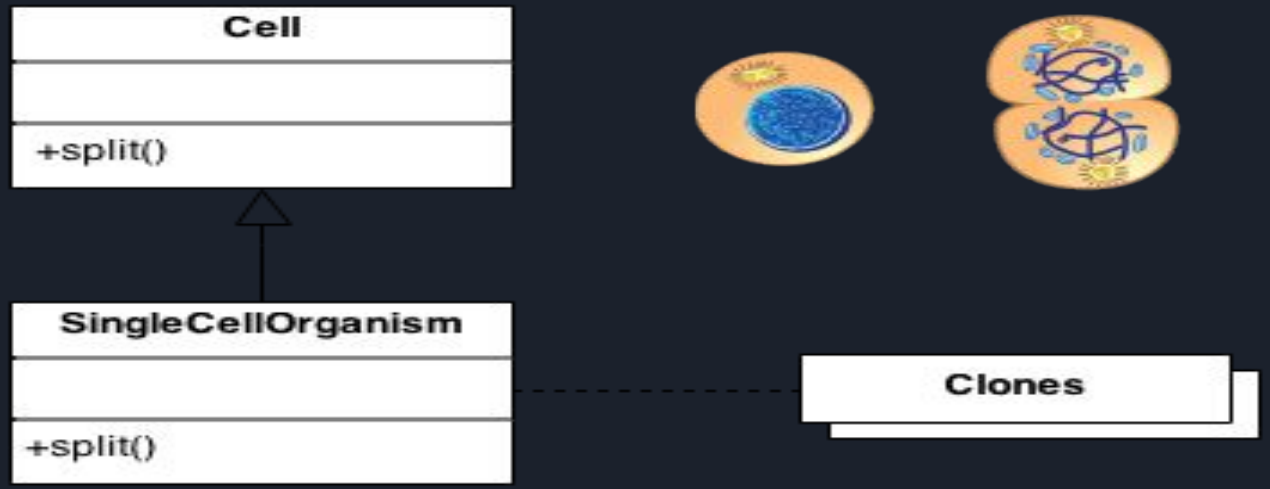
Quando usar

- Quando as classes a instanciar forem especificadas em tempo de execução.
- Para evitar a construção de uma hierarquia de classes de fábricas paralela à hierarquia de classes de produto;
- Quando as instâncias de uma classe possam ter uma dentre poucas combinações diferentes de estados. Pode ser mais conveniente instalar um número correspondente de protótipos e cloná-los, ao invés de instanciar a classe manualmente, cada vez com um estado apropriado.

Conhecendo e aplicando o padrão prototype

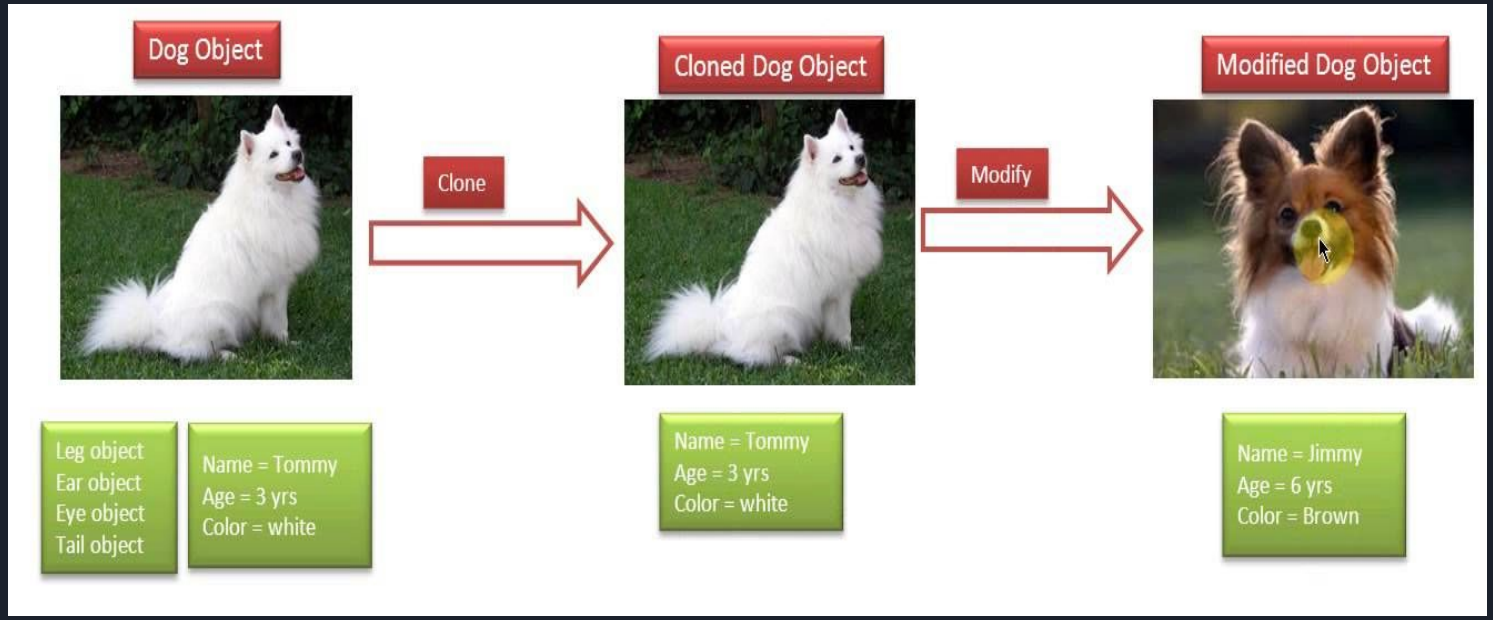
Ideia:

Criar um objeto novo, mas aproveitar o estado previamente existente em outro objeto.



Conhecendo e aplicando o padrão prototype

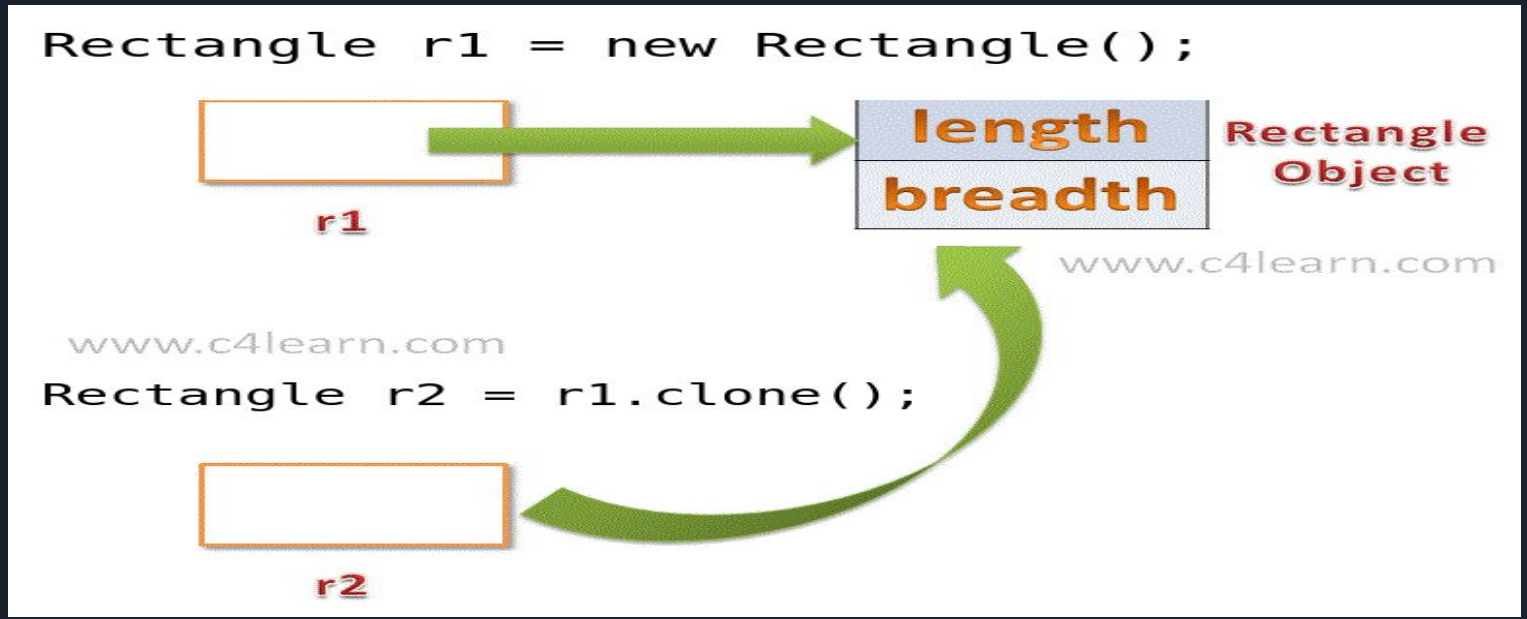
Contextualização:



Conhecendo e aplicando o padrão prototype

Exemplo:

Object.clone() é um ótimo exemplo de Prototype em Java:

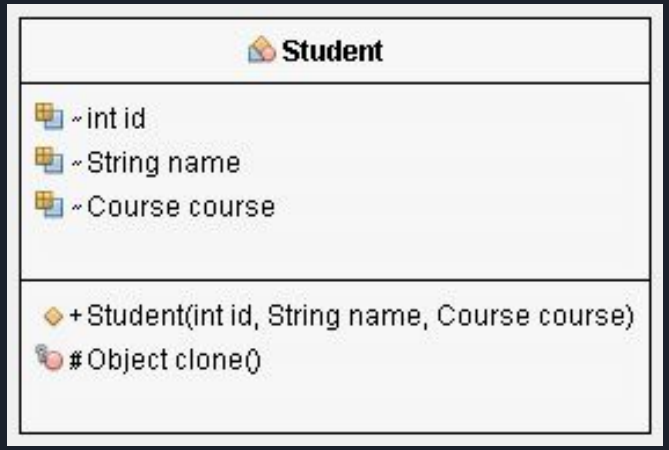


Clone superficial e clone profundo

Classe exemplo:

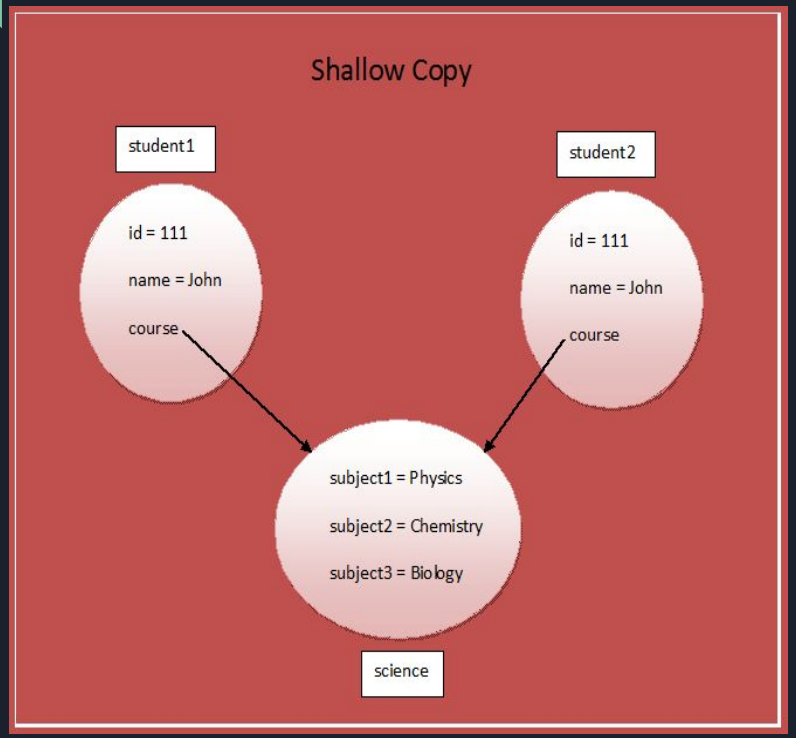
```
public class Student implements Cloneable {
    int id;
    String name;
    Course course;     //Tipo do atributo não é primitivo
    public Student(int id, String name, Course course)  {
        this.id = id;
        this.name = name;
        this.course = course;
    }

    @Override
    protected Object clone() {
        /* Apresentação dos dois
        métodos de clonagem a seguir */
    }
}
```



Clone superficial e clone profundo

Clone superficial:

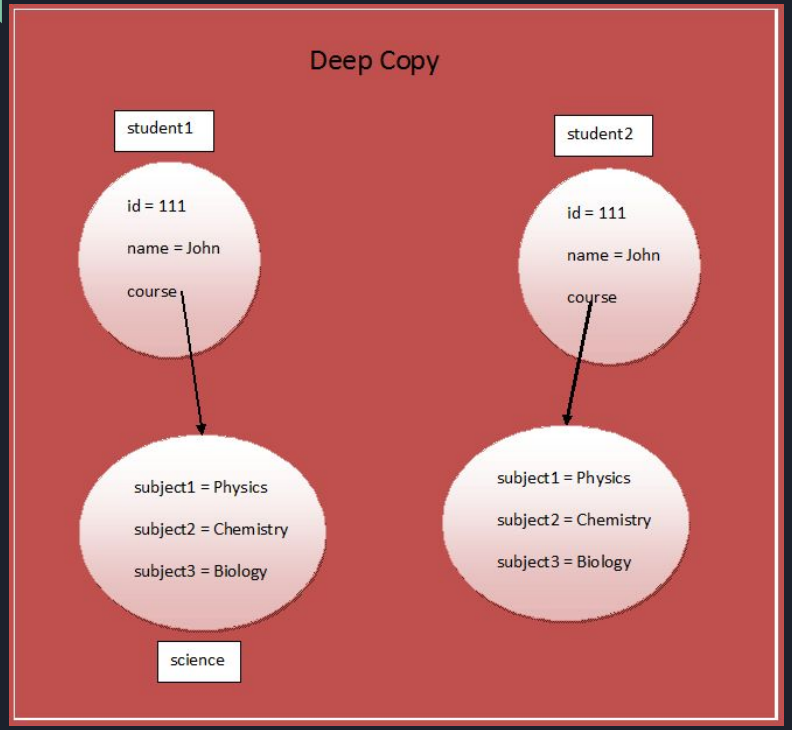


//Versão padrão do método clone(). Cria uma cópia superficial de qualquer objeto.

```
@Override
protected Object clone() {
    try {
        return super.clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
    return null;
}
```

Clone superficial e clone profundo

Clone profundo:



//Sobrescrevendo o método clone() e atribuindo clones para os atributos não primitivos, para dessa maneira realizar uma cópia profunda do objeto.

```
@Override
public Object clone () {
    try {
        Student s = (Student) super.clone();
        s.course = (Course) course.clone();
        return s;
    }catch (CloneNotSupportedException ex) {
        ex.printStackTrace();
        return null;
    }
}
```

Vantagens e Desvantagens

Vantagens:

- Poder aproveitar o estado existente de um objeto.
- Permite que um cliente crie novos objetos ao copiar objetos existentes, inclusive em tempo de execução.
- Em determinadas circunstâncias, copiar um objeto pode ser mais eficaz que criar um novo.

Desvantagens:

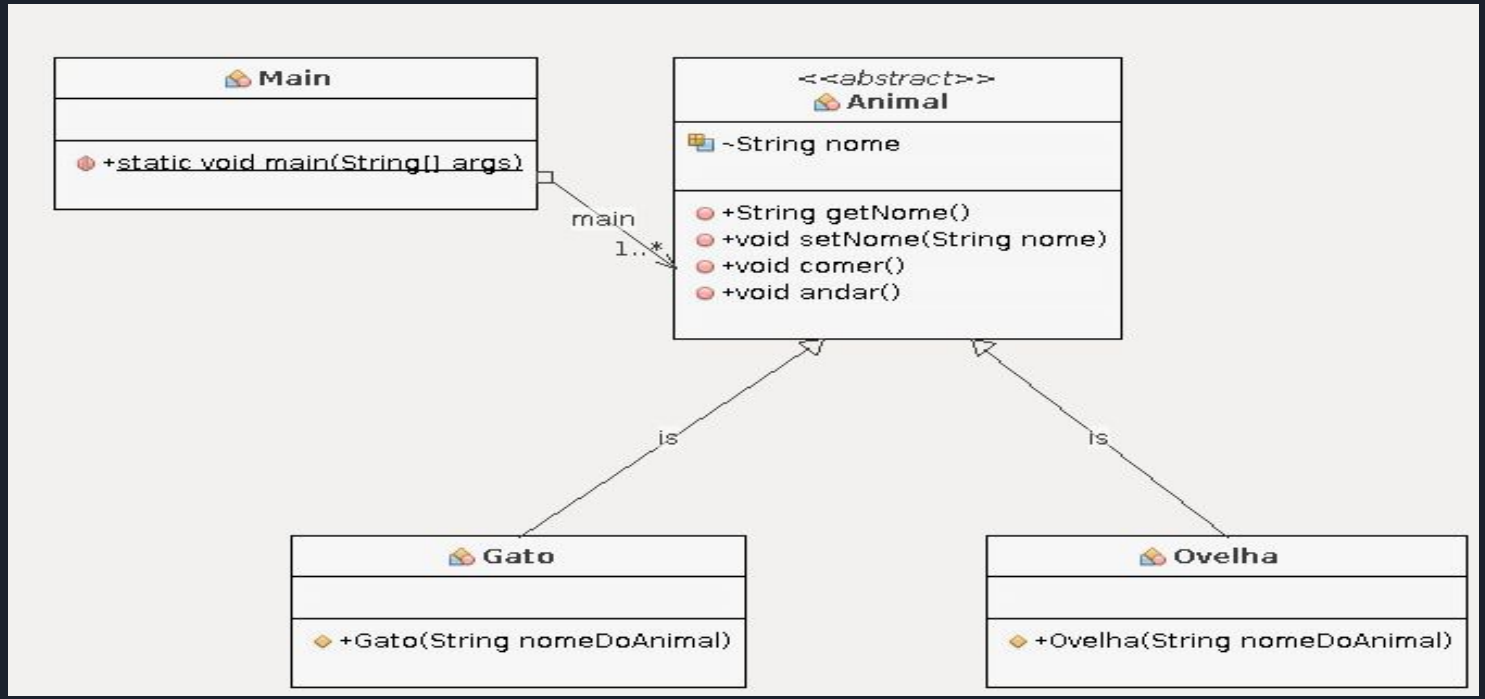
- Se torna complexo se for considerado a possibilidade de existirem referências circulares nos atributos de um objeto.
- Object.clone() pode ser usado como implementação do Prototype pattern em Java mas é preciso lembrar que ele só faz cópias rasas.
- cada subclasse de um Prototype deve implementar a operação Clone, o que pode ser difícil. Como, Por exemplo, se for utilizar um classe já existente.

Relações com outros padrões

- Basicamente o padrão prototype pode ser implementado em conjunto com quaisquer outros padrões, levando em consideração que o padrão consiste de uma especificação de “Contrato de clonagem” da classe.
- Prototype e Abstract Factory são padrões que competem entre si em várias situações. Porém eles também podem ser usados em conjunto. Um Abstract Factory pode armazenar um conjunto de protótipos a partir dos quais podem ser clonados e retornados como objetos-produto.
- Projetos que utilizam intensamente os padrões Composite e Decorator, também podem se beneficiar do uso do prototype.

Exercício

Implemente o seguinte diagrama UML utilizando o padrão prototype, após, crie um objeto do tipo Gato e um objeto do tipo Ovelha e então realize 50 clonagens de cada.



Referências Abstract Factory

https://brizeno.wordpress.com/category/padroes-de-projeto/abstract-factory/

https://www.thiengo.com.br/padrao-de-projeto-abstract-factory

http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/pat/abstractfactory.htm

https://www.devmedia.com.br/padrao-abstract-factory/23030

https://edisciplinas.usp.br/pluginfile.php/2316238/mod_resource/content/1/Aula16_StrategyAbstractFactory.pdf

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Referências Prototype

https://brizeno.wordpress.com/category/padroes-de-projeto/prototype/

https://www.devmedia.com.br/padroes-de-projeto-em-net-prototype/4597

https://pt.wikipedia.org/wiki/Prototype

https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm

https://www.devmedia.com.br/implementando-padroes-criacionais-em-java/34185

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.