Aluno: Rodrigo de Oliveira Ribeiro - 201811631

Universidade Federal de Goiás

Instituto de Informática

Questões da Prova 2

Questão 01. Manual:

Estrutura do Manual

Este manual está dividido em seis seções principais:

1. **Introdução ao GitHub:** Visão geral do que é o GitHub, sua história, evolução, principais funcionalidades e benefícios.
2. **Configuração Inicial:** Orientações sobre como criar uma conta no GitHub, instalar e configurar o Git, e os primeiros passos com repositórios.
3. **Comandos Básicos do Git:** Explicação sobre a estrutura de um repositório Git, como iniciar um repositório e os principais comandos do Git.
4. **Trabalho Colaborativo:** Como clonar e forkar repositórios, criar e gerenciar pull requests, revisar código e resolver conflitos.
5. **Funcionalidades Avançadas:** Introdução ao GitHub Actions, gerenciamento de tarefas com Issues e Projects, criação de sites estáticos com GitHub Pages e uso de integrações e APIs.
6. **Boas Práticas e Dicas:** Diretrizes para escrever bons commits, estruturar repositórios, gerenciar segurança e permissões, e usar templates e arquivos de configuração.

1. Introdução ao GitHub

O que é GitHub?

GitHub é uma plataforma de hospedagem de código-fonte com controle de versão usando o Git. Permite que desenvolvedores trabalhem juntos em projetos de software, contribuindo com código, rastreando mudanças e gerenciando tarefas.

História e Evolução do GitHub

GitHub foi lançado em 2008 por Tom Preston-Werner, Chris Wanstrath, PJ Hyett e Scott Chacon. Desde então, tornou-se uma das plataformas mais populares para desenvolvimento colaborativo de software, atraindo milhões de usuários e projetos. Em 2018, a Microsoft adquiriu o GitHub, mantendo seu foco em proporcionar uma ferramenta robusta para desenvolvedores.

Principais Funcionalidades e Benefícios

- Controle de Versão: Acompanhe e gerencie alterações no código.
- Colaboração: Facilita o trabalho em equipe através de pull requests e revisões de código.
- Automação: GitHub Actions permite automatizar fluxos de trabalho de desenvolvimento.
- Gerenciamento de Projetos: Issues e Projects ajudam a gerenciar tarefas e acompanhar o progresso.
- Publicação de Sites: GitHub Pages permite criar e hospedar sites estáticos diretamente a partir de um repositório.

---

2. Configuração Inicial

Criando uma Conta no GitHub

1. Acessar o Site do GitHub
    - Visite github.com e clique em "Sign up" no canto superior direito.
2. Preencher Informações
    - Insira seu nome de usuário, e-mail e senha. Complete o captcha e clique em "Create account".
3. Verificar E-mail
    - Confirme seu endereço de e-mail clicando no link de verificação enviado pela GitHub.

Exercício 1: Criação de Conta

- Crie uma conta no GitHub se ainda não tiver uma.

Instalando o Git e Configurando no GitHub

Instalando o Git

Windows: Baixe o instalador do Git de git-scm.com e siga as instruções.

Mac: Use Homebrew (brew install git) ou baixe diretamente de git-scm.com.

Linux: Use o gerenciador de pacotes da sua distribuição (sudo apt-get install git para Debian/Ubuntu, sudo yum install git para Fedora/Red Hat).

Configurando o Git

Abra o terminal ou prompt de comando e configure seu nome de usuário e e-mail:

git config --global user.name "Seu Nome"

git config --global user.email "seuemail@exemplo.com"

Exercício 2: Configuração do Git

- Instale o Git e configure seu nome de usuário e e-mail.

Primeiros Passos: Criando e Clonando Repositórios

1. **Criando um Repositório no GitHub**

   No GitHub, clique no botão "New" na página de repositórios.

   Preencha o nome do repositório, adicione uma descrição opcional, escolha a visibilidade (público ou privado) e clique em "Create repository".

2. **Clonando um Repositório**

   Copie a URL do repositório (HTTPS, SSH ou GitHub CLI).

   No terminal, navegue até o diretório onde deseja clonar o repositório e execute:

git clone https://github.com/usuario/repositorio.git

Exercício 3: Criando e Clonando Repositórios

- Crie um novo repositório no GitHub.
- Clone o repositório recém-criado para o seu computador.

---

3. Comandos Básicos do Git

Estrutura de um Repositório Git

Um repositório Git é um diretório especial que contém um conjunto de arquivos e pastas, além de uma subpasta .git onde o Git armazena todos os metadados e a base de dados do projeto. Ele armazena o histórico completo de todas as mudanças feitas no projeto.

Iniciando um Repositório

1. **Criando um Repositório Localmente**

   Navegue até o diretório do projeto no terminal e execute:

git init

- Isso inicializa um novo repositório Git no diretório atual.

2. **Adicionando Arquivos ao Repositório**

- Adicione arquivos ao índice (staging area) para serem rastreados pelo Git:

git add <arquivo>

- Para adicionar todos os arquivos:

git add .

Principais Comandos do Git

1. **git init**
   - Inicializa um novo repositório Git:

git init

2. **git add**

Adiciona arquivos à área de preparação (staging area).

git add <arquivo>

Para adicionar todos os arquivos:

git add .

3. **git commit**

Confirma as mudanças adicionadas à área de preparação, criando um novo snapshot do repositório:

git commit -m "Mensagem descritiva"

4. **git push**

Envia os commits locais para um repositório remoto.

git push origin <branch>

5. **git pull**

Atualiza o repositório local com as mudanças do repositório remoto.

git pull origin <branch>

Gerenciamento de Branches

Branches permitem que você trabalhe em diferentes versões do repositório ao mesmo tempo.

1. **Criar uma nova branch**
    - Para criar uma nova branch:

git checkout -b <nome-da-branch>

Alternar entre branches

- Para mudar para uma branch existente:

git checkout <nome-da-branch>

Mesclar branches

- Para mesclar uma branch com a branch atual:

git merge <nome-da-branch>

Exercício 4: Comandos Básicos do Git

- Crie um novo repositório local.
- Adicione e confirme alguns arquivos.
- Crie uma nova branch e alterne entre as branches.
- Faça um push e pull para/da branch remota.

---

4. Trabalho Colaborativo

Clonando e Forkeando Repositórios

1. **Clonando um Repositório**
    - Copie a URL do repositório (HTTPS, SSH ou GitHub CLI).
    - No terminal, navegue até o diretório onde deseja clonar o repositório e execute:

git clone https://github.com/usuario/repositorio.git

2. **Forkeando um Repositório**
    - No GitHub, vá até o repositório que deseja forkar.
    - Clique no botão "Fork" no canto superior direito.
    - Isso criará uma cópia do repositório no seu próprio perfil.

Pull Requests: Como Criar e Gerenciar

1. **Criar um Pull Request**

- Faça um fork do repositório e clone-o.
- Crie uma nova branch e faça suas mudanças.
- Empurre as mudanças para o seu repositório fork.
- No GitHub, vá até a página do repositório original e clique em "New Pull Request".

2. **Gerenciar Pull Requests**
    - Revise as mudanças propostas.
    - Comente sobre as mudanças e peça ajustes, se necessário.
    - Faça o merge do pull request se estiver satisfeito com as mudanças.

Revisão de Código e Merge de Pull Requests

1. **Revisão de Código**
    - Acesse a aba "Pull requests" no GitHub.
    - Clique no pull request para revisar.
    - Utilize comentários para discutir as mudanças propostas.
2. **Merge de Pull Requests**
    - Após a revisão e aprovação, clique no botão "Merge pull request".
    - Escolha a opção de merge desejada (Merge Commit, Squash and Merge, Rebase and Merge).

Resolvendo Conflitos

1. **Identificar Conflitos**
    - Se houver conflitos, o Git notificará durante o pull ou merge.
2. **Resolver Conflitos**
    - Abra os arquivos com conflitos e edite manualmente para resolver as diferenças.
    - Após resolver, adicione os arquivos corrigidos:

git add <arquivo>

- Finalize com um commit

git commit

Exercício 5: Trabalho Colaborativo

- Faça o fork de um repositório.
- Crie um pull request com algumas mudanças.
- Revise e faça o merge do pull request.
- Resolva um conflito simples em um pull request.

---

5. Funcionalidades Avançadas

GitHub Actions: Automatizando Fluxos de Trabalho

GitHub Actions permite automatizar fluxos de trabalho de desenvolvimento de software diretamente no repositório GitHub. Com Actions, você pode construir, testar e implantar seu código sempre que ocorrer um push ou pull request.

1. **Criando um Workflow**
    - No repositório, crie uma pasta .github/workflows.
    - Dentro dessa pasta, crie um arquivo YAML (ex: ci.yml) com o conteúdo:

```
name: CI
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'
    - name: Install dependencies
      run: npm install
```

- name: Run tests

run: npm test

2. **Executando o Workflow**
    - Faça um commit e push das alterações.
    - Navegue até a aba "Actions" no repositório para visualizar o status e resultados do workflow.

Exercício 6: GitHub Actions

- Crie um workflow simples que execute testes automatizados para um projeto Node.js.

Issues e Projects: Gerenciamento de Tarefas e Projetos

1. **Criando Issues**
    - Vá até a aba "Issues" no repositório e clique em "New issue".
    - Preencha o título e a descrição da issue e clique em "Submit new issue".
2. **Gerenciando Issues**
    - Atribua issues a colaboradores, adicione labels e milestones para organização.
3. **Criando Projects**
    - Vá até a aba "Projects" e clique em "New project".
    - Escolha um template ou crie um projeto em branco.
    - Adicione colunas e cards para organizar as tarefas.

Exercício 7: Issues e Projects

- Crie uma issue para um bug ou nova feature.
- Crie um project board e adicione a issue criada a ele.

GitHub Pages: Criando Sites Estáticos com GitHub

GitHub Pages permite que você crie sites estáticos diretamente a partir de um repositório GitHub.

1. **Habilitando GitHub Pages**
    - Vá até as configurações do repositório.
    - Na seção "GitHub Pages", selecione a branch a partir da qual o site será gerado (geralmente main ou gh-pages).

- Clique em "Save".

2. **Publicando um Site**
    - Adicione um arquivo index.html à branch selecionada.
    - Navegue até https://<seu-usuario>.github.io/<seu-repositorio> para visualizar o site.

Exercício 8: GitHub Pages

- Habilite GitHub Pages para um repositório e publique um site simples.

Integrações e APIs

1. **Integrando com Outros Serviços**
    - No GitHub Marketplace, explore as integrações disponíveis, como CI/CD, monitoramento, etc.
    - Instale e configure as integrações conforme necessário.
2. **Utilizando a API do GitHub**
    - Acesse a documentação da API do GitHub para aprender sobre os endpoints disponíveis.
    - Exemplos de uso da API com cURL:

curl -H "Authorization: token <seu-token>"
https://api.github.com/repos/usuario/repositorio

Exercício 9: Integrações e APIs

- Integre um serviço de CI/CD ao seu repositório.
- Faça uma chamada simples à API do GitHub para listar os repositórios do seu usuário.

---

6. Boas Práticas e Dicas

Escrevendo Bons Commits e Mensagens

1. **Estrutura de uma Boa Mensagem de Commit**
    - Linha de assunto: Curta e descritiva (máximo de 50 caracteres).

- Linha em branco.
- Corpo da mensagem: Detalhes sobre as mudanças (máximo de 72 caracteres por linha).

2. **Exemplo de Boa Mensagem de Commit**

Corrige bug na função de login

- Corrige a validação de senhas com caracteres especiais

- Adiciona testes unitários para cobrir os novos casos de uso

Exercício 10: Boas Mensagens de Commit

- Faça commits com mensagens claras e estruturadas em um repositório.

Estrutura Organizacional de Repositórios

1. **Organizando Arquivos e Pastas**
   - Mantenha uma estrutura de diretórios clara e lógica.
   - ***Exemplos:***

├── src/

├── tests/

├── docs/

└── assets/

2. **Documentação**
   - Adicione um arquivo README.md com instruções claras sobre o projeto.
   - Use CONTRIBUTING.md para guias de contribuição e CODE_OF_CONDUCT.md para regras de conduta.

Exercício 11: Estrutura de Repositório

- Organize um repositório existente, adicionando uma estrutura clara e documentação.

Segurança e Permissões

1. **Gerenciando Colaboradores**

- Vá até as configurações do repositório.
- Na seção "Manage access", adicione ou remova colaboradores e ajuste permissões.

2. **Utilizando Secrets**
    - No GitHub, vá até as configurações do repositório.
    - Na seção "Secrets", adicione segredos (como tokens de API) que podem ser usados em workflows.

Exercício 12: Segurança e Permissões

- Adicione um colaborador a um repositório e configure permissões adequadas.
- Adicione um segredo ao repositório e utilize-o em um workflow do GitHub Actions.

Uso de Templates e Arquivos de Configuração

1. **Criando Templates**
    - Use templates para issues e pull requests criando um diretório .github/ISSUE_TEMPLATE/ e .github/PULL_REQUEST_TEMPLATE/.
2. **Arquivo .gitignore**
    - Adicione um arquivo .gitignore para especificar quais arquivos ou diretórios não devem ser rastreados pelo Git.
    - Exemplo de um .gitignore:

 # Logs

****.log***

 # Node modules

node_modules/

 # Build output

dist/

3. **Arquivo README.md**
    - Forneça uma visão geral do projeto, instruções de instalação, uso e contribuição.

Exercício 13: Templates e Arquivos de Configuração

- Adicione templates de issue e pull request a um repositório.
- Crie um arquivo .gitignore adequado para um projeto de sua escolha.
- Melhore o README.md de um repositório com informações detalhadas.

Questão 02. Utilizando o teste estrutural (também conhecido como teste de caixa branca), responda o que se pede a seguir:

a) Qual o grafo de fluxo de controle para este código?

Descrição textual do grafo de fluxo de controle:

1. **Início**: Começa a execução do programa.
2. **Chamado para main()**:
   - numero é definido como 29.
   - Chama a função ehPrimo(numero).
3. **Dentro da função ehPrimo()**:
   - Verifica se num <= 1 (nó de decisão):
     - **Verdadeiro**: Retorna false.
     - **Falso**: Continua para o próximo passo.
   - Loop for (int i = 2; i * i <= num; i++):
     - **Verdadeiro**: Verifica num % i == 0 (nó de decisão):
       - **Verdadeiro**: Retorna false.
       - **Falso**: Continua para o próximo valor de i.
     - **Falso**: Sai do loop e retorna true.
4. **Volta para main()**:
   - Verifica o resultado de ehPrimo(numero):
     - **Verdadeiro**: Imprime "%d é primo".
     - **Falso**: Imprime "%d não é primo".

5. **Fim**: Termina a execução do programa.

b) Quantos caminhos independentes existem neste código?

Complexidade Ciclomática = E−N+2

onde E é o número de arestas e N é o número de nós no grafo de fluxo de controle.

Nós (N): 8 (Início, 4 decisões, 2 retornos, 1 fim)

Arestas (E): 9 (Incluindo todos os caminhos de decisão e loop)

Complexidade Ciclomática=9−8+2=3Complexidade Ciclomática = 9−8+2 = 3

Então, existem 3 caminhos independentes.

c) Liste todos os caminhos independentes identificados.

Caminho 1: Entrada → num <= 1 (falso) → Loop (primeira iteração, i=2) → num % i == 0 (falso) → Loop (saída) → Retorna true.

Caminho 2: Entrada → num <= 1 (verdadeiro) → Retorna false.

Caminho 3: Entrada → num <= 1 (falso) → Loop (primeira iteração, i=2) → num % i == 0 (verdadeiro) → Retorna false.

d) Para cada caminho independente, descreva um caso de teste que garantiria a cobertura desse caminho.

Caminho 1:

- Caso de Teste: numero = 29
- Descrição: O número 29 é primo, então a função deve retornar true após iterar pelo loop sem encontrar divisores.

Caminho 2:

- Caso de Teste: numero = 1
- Descrição: O número 1 não é primo, a função deve retornar false imediatamente.

Caminho 3:

- Caso de Teste: numero = 4
- Descrição: O número 4 não é primo, a função deve retornar false na primeira iteração do loop quando i = 2.

e) Quais são as condições lógicas presentes no código?

Condicional 1: if (num <= 1)

Condicional 2: if (num % i == 0) dentro do loop for (int i = 2; i * i <= num; i++)

f) Descreva um conjunto mínimo de casos de teste que garantam a cobertura de todas as condições lógicas.

Caso de Teste 1: numero = 1

- **Cobertura**: num <= 1 (verdadeira)

Caso de Teste 2: numero = 2

- **Cobertura**: num <= 1 (falsa) e num % i == 0 (falsa para i = 2)

Caso de Teste 3: numero = 4

- **Cobertura**: num <= 1 (falsa) e num % i == 0 (verdadeira para i = 2)

Caso de Teste 4: numero = 29

- **Cobertura**: num <= 1 (falsa) e num % i == 0 (falsa para todos i testados no loop)

g) Descreva os casos de teste usando análise de valor limite considerando que um número primo é aquele que é maior que 1 é divisível apenas por 1 e por ele mesmo.

Caso de Teste 1: numero = 0

- **Justificativa**: Valor limite abaixo de 1 (não é primo).
- A função deve retornar false.

Caso de Teste 2: numero = 1

- **Justificativa**: Valor limite igual a 1 (não é primo).
- A função deve retornar false.

Caso de Teste 3: numero = 2

- **Justificativa**: Menor número primo.
- A função deve retornar true.

Caso de Teste 4: numero = 3

- **Justificativa**: Próximo número primo após o menor.
- A função deve retornar true.

Caso de Teste 5: numero = 4

- **Justificativa**: Menor número não primo maior que 2.
- A função deve retornar false.

Caso de Teste 6: numero = 5

- **Justificativa**: Número primo logo após o primeiro número não primo maior que 2.
- A função deve retornar true.

Caso de Teste 7: numero = 28

- **Justificativa**: Número composto próximo a um número primo (29).
- A função deve retornar false.

Caso de Teste 8: numero = 29

- **Justificativa**: Número primo próximo a um número composto (28).
- A função deve retornar true.

Questão 03.

A) No contexto de gerenciamento de projetos de software, explique o processo de análise de riscos.

A análise de riscos é um processo crítico no gerenciamento de projetos de software, que envolve identificar, avaliar e priorizar possíveis riscos que podem afetar o sucesso do projeto.

1. **Identificação dos Riscos:**

- Fontes de Riscos: Identificar possíveis fontes de riscos, como requisitos incompletos, mudanças de escopo, falhas técnicas, atrasos no cronograma, falta de recursos, entre outros.
- Listagem de Riscos: Criar uma lista abrangente de todos os riscos possíveis, muitas vezes utilizando técnicas como brainstorming, entrevistas, e análise de documentação.

2. **Análise Qualitativa dos Riscos:**
   - Probabilidade e Impacto: Avaliar a probabilidade de ocorrência de cada risco e o impacto potencial no projeto.
   - Classificação dos Riscos: Classificar os riscos com base na combinação de sua probabilidade e impacto, geralmente em categorias como baixo, médio e alto.
3. **Análise Quantitativa dos Riscos:**
   - Modelagem e Simulação: Utilizar técnicas quantitativas, como análise de Monte Carlo, para estimar numericamente o impacto dos riscos no cronograma e no orçamento do projeto.
   - Priorização: Priorizar os riscos com base na análise quantitativa para focar naqueles que têm maior potencial de impacto.
4. **Planejamento de Respostas aos Riscos:**
   - Estratégias de Mitigação: Desenvolver estratégias para reduzir a probabilidade ou impacto dos riscos, como adicionar buffers de tempo, melhorar a comunicação ou treinar a equipe.
   - Planos de Contingência: Preparar planos de ação para serem executados caso os riscos se materializem.
5. **Monitoramento e Controle dos Riscos:**
   - Revisão Contínua: Monitorar continuamente os riscos durante todo o ciclo de vida do projeto e ajustar as estratégias conforme necessário.
   - Relatórios de Riscos: Manter registros atualizados dos riscos e das ações tomadas, e comunicar regularmente o status dos riscos às partes interessadas.

B) Uma empresa de desenvolvimento de software de médio porte precisa desenvolver um software de vendas de pacotes de viagens para uma companhia de turismo. Explique

como XP e Scrum podem ser combinados por esta empresa no desenvolvimento de software.

XP (Extreme Programming) e Scrum são duas metodologias ágeis que podem ser combinadas para aproveitar os pontos fortes de ambas. Veja como uma empresa de desenvolvimento de software de médio porte pode combiná-las para desenvolver um software de vendas de pacotes de viagens para uma companhia de turismo:

1. Estrutura de Scrum:

- **Papéis:**
    - Product Owner: Representa os interesses da companhia de turismo e define as prioridades do backlog.
    - Scrum Master: Facilita o processo Scrum e remove impedimentos.
    - Equipe de Desenvolvimento: Equipe multifuncional que trabalha na entrega do software.
- **Eventos:**
    - Sprints: Ciclos de desenvolvimento curtos e repetitivos, geralmente de 2 a 4 semanas.
    - Planejamento da Sprint: Definição do trabalho a ser realizado durante a sprint.
    - Daily Stand-ups: Reuniões diárias para acompanhar o progresso e discutir impedimentos.
    - Revisão da Sprint: Demonstração do trabalho realizado no final de cada sprint.
    - Retrospectiva da Sprint: Reflexão sobre o que foi bem e o que pode ser melhorado no próximo ciclo.
- **Artefatos:**
    - Product Backlog: Lista priorizada de requisitos e funcionalidades.
    - Sprint Backlog: Conjunto de itens do Product Backlog selecionados para a sprint.
    - Incremento: Versão funcional do software ao final de cada sprint.

2. Práticas de XP:

- Desenvolvimento Orientado a Testes (TDD): Escrever testes antes de desenvolver a funcionalidade para garantir a qualidade e a funcionalidade do software.
- Programação em Pares: Dois desenvolvedores trabalham juntos no mesmo código, revisando o trabalho um do outro em tempo real.
- Integração Contínua: Integrar e testar o código frequentemente para detectar problemas cedo.
- Refatoração: Melhorar continuamente o código para torná-lo mais eficiente e fácil de manter.
- Planejamento de Releases: Pequenas entregas frequentes para proporcionar valor contínuo ao cliente.

Como Combinar XP e Scrum:

- Sprints e Entregas Incrementais: Use a estrutura de sprints de Scrum para organizar o trabalho em ciclos curtos e incrementais, enquanto as práticas de XP garantem a qualidade do código dentro de cada sprint.
- Eventos Scrum com Práticas de XP: Realize eventos Scrum regulares (Daily Stand-ups, Revisões de Sprint, Retrospectivas) e incorpore práticas de XP como programação em pares e TDD durante as sprints.
- Foco na Qualidade e Colaboração: XP enfatiza a qualidade técnica e a colaboração intensa, o que complementa a abordagem iterativa e incremental de Scrum.

Exemplo de Integração:

1. **Início do Projeto:**
    - Realize uma reunião de planejamento inicial com o Product Owner para criar um Product Backlog.
    - Defina a duração das sprints (por exemplo, 2 semanas).
2. **Durante a Sprint:**
    - A equipe trabalha em pares (programação em pares) e pratica TDD para garantir que o código atenda aos requisitos.
    - Integração contínua é usada para manter o código sempre em um estado funcional.
3. **Revisão e Retrospectiva:**

- Ao final de cada sprint, a equipe demonstra o software funcional ao Product Owner e às partes interessadas.
- Durante a retrospectiva, a equipe discute o que funcionou bem e o que precisa ser melhorado, tanto em termos de processo quanto de práticas de desenvolvimento.

Ao combinar XP e Scrum, a empresa pode desenvolver o software de vendas de pacotes de viagens de maneira eficiente, mantendo um foco na entrega contínua de valor e na qualidade técnica do produto.

Questão 4. O que são padrões de projeto em engenharia de software?

Os padrões de projeto são soluções generalizadas para problemas recorrentes que surgem durante o desenvolvimento de software. Eles não são pedaços de código reutilizáveis, mas sim descrições abstratas de soluções que podem ser aplicadas em vários contextos. Os padrões de projeto ajudam os desenvolvedores a criar software de forma mais eficiente, promovendo a reutilização de soluções testadas e comprovadas.

Os padrões de projeto são categorizados em três tipos principais:

1. **Padrões Criacionais**: Envolvem a criação de objetos, abstraindo a instância do objeto do seu uso.

Exemplos: Singleton, Factory Method, Abstract Factory, Builder, Prototype.

2. **Padrões Estruturais**: Lidam com a composição de classes ou objetos.

   Exemplos: Adapter, Composite, Proxy, Flyweight, Facade, Bridge, Decorator.

3. **Padrões Comportamentais**: Focam na comunicação entre objetos.

   Exemplos: Strategy, Observer, Command, Chain of Responsibility, State, Mediator, Memento, Visitor, Iterator, Template Method.

Explique como os padrões de projeto estão relacionados com a atividade de Evolução de Software.

A evolução de software envolve a modificação contínua de um sistema de software após sua entrega inicial para corrigir falhas, melhorar o desempenho ou outros atributos, ou adaptar o software a um ambiente em mudança. Padrões de projeto são altamente relevantes nessa atividade porque:

1. Facilitam a Manutenção: Padrões de projeto fornecem uma estrutura clara e bem definida para o código, tornando-o mais fácil de entender, modificar e manter.
2. Promovem a Reutilização: Ao utilizar padrões de projeto, os desenvolvedores podem reutilizar soluções comprovadas, o que pode reduzir o tempo e o esforço necessários para implementar mudanças.
3. Aprimoram a Flexibilidade: Muitos padrões de projeto são projetados para promover a flexibilidade e extensibilidade do software, facilitando a adição de novas funcionalidades sem grandes refatorações.
4. Reduzem a Complexidade: Ao aplicar padrões de projeto, os desenvolvedores podem modularizar o código, dividindo problemas complexos em partes mais gerenciáveis.

Explique o padrão de projeto Strategy.

O padrão de projeto Strategy é um padrão comportamental que permite que uma família de algoritmos seja definida, encapsulada, e tornada intercambiável. O padrão Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam.

Estrutura do Strategy

- Contexto: Uma classe que usa uma Strategy. Mantém uma referência a um objeto Strategy e pode alterar essa referência para mudar o comportamento.
- Strategy (interface): Interface comum para todas as estratégias, definindo um método que cada estratégia deve implementar.
- ConcreteStrategy: Implementações concretas da interface Strategy. Cada ConcreteStrategy implementa o algoritmo de uma maneira específica.

Exemplo do Strategy

Imagine que estamos desenvolvendo um software de cálculo de preços para um e-commerce. Dependendo da época do ano ou da política de marketing, diferentes estratégias de cálculo de descontos podem ser aplicadas.

Estrutura do Código:

1. **Strategy Interface:**

```
public interface DiscountStrategy {
    double applyDiscount(double price);
}
```

2. **Concrete Strategies:**

```
public class NoDiscountStrategy implements DiscountStrategy {
    @Override
    public double applyDiscount(double price) {
        return price;
    }
}
public class ChristmasDiscountStrategy implements DiscountStrategy {
```

```
    @Override
    public double applyDiscount(double price) {
        return price * 0.9; // 10% de desconto
    }
}
public class BlackFridayDiscountStrategy implements DiscountStrategy {
    @Override
    public double applyDiscount(double price) {
        return price * 0.7; // 30% de desconto
    }
}
```

3. **Context:**

```
public class PriceCalculator {
    private DiscountStrategy discountStrategy;

    public void setDiscountStrategy(DiscountStrategy discountStrategy) {
        this.discountStrategy = discountStrategy;
    }

    public double calculatePrice(double price) {
        return discountStrategy.applyDiscount(price);
    }
}
```

4. **Uso do Contexto:**

```
public class Main {
```

```
    public static void main(String[] args) {
        PriceCalculator calculator = new PriceCalculator();

        // Sem desconto
        calculator.setDiscountStrategy(new NoDiscountStrategy());
        System.out.println("Preço final: " + calculator.calculatePrice(100));

        // Desconto de Natal
        calculator.setDiscountStrategy(new ChristmasDiscountStrategy());
        System.out.println("Preço final: " + calculator.calculatePrice(100));

        // Desconto de Black Friday
        calculator.setDiscountStrategy(new BlackFridayDiscountStrategy());
        System.out.println("Preço final: " + calculator.calculatePrice(100));
    }
}
```

Obs:. Neste exemplo, o PriceCalculator pode mudar a estratégia de desconto em tempo de execução, dando flexibilidade e facilitando a evolução do software à medida que novos tipos de descontos são adicionados.

Referências:

https://training.github.com/downloads/pt_BR/github-git-cheat-sheet.pdf

https://docs.github.com/en/repositories/working-with-files

https://docs.github.com/en

Sommerville, Ian. Engenharia de software. 10. ed. Rio de Janeiro: LTC, 2016.