CSC207H Lecture 8

Sadia Sharmin

Nov 1, 2016

git branch

- A branch represents an independent line of development
- You can think of them as a way to request a brand new working directory, staging area, and project history
- When you want to add a new feature or fix a bug you spawn a new branch to encapsulate your changes
- Makes sure that unstable code is never committed to the main code base
- Push a branch to the central repository using `git push -u origin <branch-name>` the first time you do it; after that, just use `git push`

Source:
`https://www.atlassian.com/git/tutorials/using-branches/`

git checkout

- git checkout command lets you select which branch you want to switch to + work on
- `git checkout <existing-branch>` makes `<existing-branch>` the current branch, and updates the working directory to match
- When you want to start a new feature, you create a branch with git branch, then check it out with git checkout

Source:
`https://www.atlassian.com/git/tutorials/using-branches/`

git merge

- `git merge` Lets you merge your branches back into a single branch
- This merges your changes into the current branch; that is the current branch will be updated to reflect the merge, but the target branch will be completely unaffected
- Use `git checkout` for selecting the current branch and `git branch -d` for deleting the obsolete target branch

Source:
`https://www.atlassian.com/git/tutorials/using-branches/`

Dealing with merge conflicts

- When you encounter a merge conflict, running the `git status` command shows you which files need to be resolved
- Fix these files, then `git add`, then commit and push as usual

Source:
`https://www.atlassian.com/git/tutorials/using-branches/`

In Eclipse Git

- `git branch` :
  Team > Switch To ... > New Branch
- `git checkout <branch-name>` :
  Team > Switch To ... > `<branch-name>`
- `git push` :
  Team > Push to upstream
- `git merge` :
  Team > Merge... , then select which local or remote branch you want to merge into the current one

Design Pattern: Factory

Scenario:

- A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

Sources:

`https://sourcemaking.com/design_patterns/factory_method`

Design Pattern: Factory

What you need:
The typical implementation uses a single class with a single method (the factory method) and this method returns an object based on the input passed as an argument. Which object is it? Well, that depends on the passed parameter.

- Product: A superclass interface that defines all standard and generic behavior
- Concrete Products: Subclasses that implement this interface
- Creator: The factory that creates a Product object and returns it

Sources:
`https://sourcemaking.com/design_patterns/factory_method`
`https://iluxonchik.github.io/design-patterns-notes/`

Design Pattern: Builder

Scenario:

- The purpose of the builder pattern is to separate the construction of a complex object from its representation.
- Example: Happy meals: The happy meal typically consists of a hamburger, fries, coke and toy. No matter whether you choose different burgers/drinks, the construction of the kids meal follows the same process.

Sources: `https://dzone.com/articles/design-patterns-builder`

Design Pattern: Builder

What you need:

- Product: the object (usually a complex one) that we are creating; includes all the classes that define what we're constructing
- Builder: an interface for creating the parts that make up the Product
- Concrete Builder: keeps track of the representation it creates, returns the product
- Director: constructs the object through Builder's interface

Sources: `https://dzone.com/articles/design-patterns-builder`