**Basic Understanding Questions:**

1. **What is the Template Method Pattern?**
   * **Answer:** The Template Method Pattern defines the skeleton of an algorithm in the base class but allows subclasses to provide specific implementations for some steps of the algorithm. This ensures that the overall structure of the algorithm remains the same, but certain steps can be customized by subclasses. The template method is a method in the base class that defines the structure of the algorithm.
   * **Example:** Think of a cooking recipe: the basic structure is the same (prepare ingredients, cook, serve), but the actual steps for preparing and cooking can vary depending on the dish.
2. **Can you give an example where the Template Method Pattern would be useful in real-world applications?**
   * **Answer:** A good example is in document generation systems. Suppose there is a need to generate reports in multiple formats (e.g., PDF, Word, Excel). While the steps to generate a report (e.g., gather data, format it, save the document) are similar, the details (e.g., formatting the content for PDF or Word) can vary. The Template Method allows us to define the common steps in a base class and let the subclasses customize the formatting.
   * **Example:** ReportGenerator base class with methods like gather\_data(), format\_data(), save\_document(), and generate\_report(). The generate\_report() method would call the others in order, while subclasses override format\_data() for specific formats.
3. **What is the role of the "hook" method in the Template Method Pattern?**
   * **Answer:** Hook methods are optional methods in the base class that provide a way to "hook into" the algorithm defined in the template method. These hooks can be overridden by subclasses to add additional behavior or modify default behavior without changing the overall algorithm's flow.
   * **Example:** In a report generation system, you might have a hook like customize\_layout(). This method might have a default behavior (e.g., use a standard layout), but subclasses could override it to customize the layout for specific types of reports.
4. **What is the difference between the Template Method Pattern and the Strategy Pattern?**
   * **Answer:** The key difference between the Template Method and Strategy Pattern is the way flexibility is achieved.
     + The **Template Method Pattern** defines a fixed algorithm structure and allows subclasses to override parts of the algorithm (steps). The algorithm's skeleton remains the same across all subclasses, but some steps can be customized.
     + The **Strategy Pattern** provides a way to define different algorithms (strategies) and allows the algorithm to be chosen at runtime. The Strategy Pattern allows different strategies to be swapped in and out, whereas the Template Method relies on inheritance to define behavior.
   * **Example:** A payment system can use the Strategy Pattern to switch between different payment methods (Credit Card, PayPal), while a report generation system might use the Template Method Pattern to follow a fixed sequence of steps with customizable behavior at specific stages.
5. **How does the Template Method Pattern promote code reuse?**
   * **Answer:** The Template Method Pattern promotes code reuse by defining the common steps of an algorithm in a base class. The template method calls the steps in a specific order, and subclasses can override only the specific steps that differ. This allows code to be reused across different subclasses without needing to repeat the common parts of the algorithm.
   * **Example:** In a game, the steps to play a round might include starting the round, performing actions, and ending the round. These steps could be common, but subclasses (like for different types of games) can customize specific actions, like how players take turns.

**Practical Application Questions:**

1. **Suppose you are designing a system for generating different types of reports (e.g., PDF, Excel). How would you apply the Template Method Pattern?**
   * **Answer:** You would define an abstract base class, say ReportGenerator, which contains the template method generate\_report(). This method defines the overall steps to generate a report, like:
     1. Gather data
     2. Format data (this would be a hook method that subclasses override)
     3. Save the report
   * Subclasses like PDFReportGenerator and ExcelReportGenerator would override the format\_data() method to implement the formatting for their specific formats, while reusing the common steps for gathering and saving data.
2. **Can you implement a Template Method for a game where different characters have common behaviors but unique ways of performing actions (e.g., attacking, defending)?**
   * **Answer:** Yes, the Template Method can define a play\_turn() method with the general structure of a turn (e.g., prepare for action, perform action, finish turn). Specific actions like attack() or defend() can be customized in subclasses (e.g., Warrior or Mage).

python

Copy code

class Character:

def play\_turn(self):

self.prepare\_for\_action()

self.perform\_action()

self.finish\_turn()

def prepare\_for\_action(self):

print("Preparing for action...")

def finish\_turn(self):

print("Ending the turn...")

def perform\_action(self):

raise NotImplementedError("Subclasses must implement this method")

class Warrior(Character):

def perform\_action(self):

print("Warrior is attacking!")

class Mage(Character):

def perform\_action(self):

print("Mage is casting a spell!")

1. **Can you explain how the Template Method Pattern can be extended in Python?**
   * **Answer:** In Python, the Template Method Pattern can be implemented using inheritance and method overriding. The base class defines the algorithm and calls methods that can be overridden by subclasses. Python's dynamic nature allows for easy customization of the steps.
   * **Example:** Subclass ComputerBuilder overrides steps of building a computer in the build\_computer() method, while the overall structure remains unchanged.

**Advanced Questions:**

1. **What are the limitations of the Template Method Pattern?**
   * **Answer:**
     + **Rigid structure:** Once the algorithm skeleton is defined, it is difficult to change the flow of the algorithm. The Template Method is not flexible in accommodating new steps without modifying the base class.
     + **Code duplication:** If many subclasses override the same methods, you may end up with duplicated code.
     + **Inheritance overhead:** Since the pattern relies on inheritance, it might introduce unnecessary complexity in some cases.
2. **How can you handle exceptions in the Template Method Pattern?**

* **Answer:** Exception handling can be incorporated into the template method or individual steps of the algorithm. The template method can catch exceptions from the steps and either handle them or propagate them.
* **Example:** If a specific step in report generation fails, you can catch and handle the exception in the template method to ensure the remaining steps are still executed, or you can let it propagate to the caller.

1. **How does the Template Method Pattern fit into the overall concept of design patterns and object-oriented design principles?**

* **Answer:** The Template Method Pattern fits into **object-oriented principles** like **inheritance** (subclasses override specific parts of the algorithm) and **polymorphism** (subclasses provide their own implementations). It is useful for **code reuse** because it allows common behavior to be implemented once in the base class while allowing subclasses to modify specific steps.

1. **In what situations would you prefer the Template Method Pattern over other behavioral design patterns like State or Strategy?**

* **Answer:** The Template Method Pattern is suitable when:
  + The sequence of steps in an algorithm is fixed, but some steps can be customized by subclasses.
  + You want to ensure that certain steps are executed in a specific order, but allow subclasses to implement them differently.
  + The **Strategy Pattern** would be more appropriate if you need to vary the entire algorithm at runtime, and the **State Pattern** would be useful when the state of an object changes its behavior.

**Coding Exercise Questions:**

1. **Implement a Template Method Pattern for preparing a dish where each step is common but the way each dish is prepared varies (e.g., preparing pasta, salad, etc.).**

* **Answer:**

python

Copy code

class Dish:

def prepare\_dish(self):

self.prepare\_ingredients()

self.cook()

self.serve()

def prepare\_ingredients(self):

print("Preparing ingredients...")

def cook(self):

raise NotImplementedError("Subclasses must implement this step")

def serve(self):

print("Serving the dish...")

class Pasta(Dish):

def cook(self):

print("Boiling the pasta...")

class Salad(Dish):

def cook(self):

print("Mixing the vegetables...")

pasta = Pasta()

pasta.prepare\_dish() # Output: Preparing ingredients... Boiling the pasta... Serving the dish...

salad = Salad()

salad.prepare\_dish() # Output: Preparing ingredients... Mixing the vegetables... Serving the dish...

1. **You are given an abstract class Document with methods to read(), process(), and save(). Implement a Template Method that defines a skeleton of the process method for different types of documents (e.g., Word, PDF).**

* **Answer:**

python

Copy code

class Document:

def process\_document(self):

self.read()

self.process()

self.save()

def read(self):

print("Reading the document...")

def process(self):

raise NotImplementedError("Subclasses must implement this method")

def save(self):

print("Saving the document...")

class WordDocument(Document):

def process(self):

print("Processing Word document...")

class PDFDocument(Document):

def process(self):

print("Processing PDF document...")

word\_doc = WordDocument()

word\_doc.process\_document() # Output: Reading the document... Processing Word document... Saving the document...

pdf\_doc = PDFDocument()

pdf\_doc.process\_document() # Output: Reading the document... Processing PDF document... Saving the document...