Certainly, based on the sources and our conversation, here are four quiz-style questions categorized as "Memorization":

**Question 1: Memorization (ghi nhớ)**

According to the provided source code snippets for classes like Knight and Warrior, which specific Java annotation is used just before the method signature of public double getCombatScore() to indicate that it overrides a method from a superclass or interface?

a) @Abstract b) @Override c) @Implement d) @Signature

**Question 2: Memorization (ghi nhớ)**

Based on the assignment specification, which specific classes are explicitly listed as being included in "Part A" and provided to you in bytecode form (.class files)?

a) Knight, Warrior, Paladin, DeathEater b) Combatable, Human, Fighter, Monster, Main, Battle c) TeamMaker, Complex, Utility d) Main, Battle, TeamMaker

**Question 3: Memorization (ghi nhớ)**

According to the assignment specification, what specific diagram is referenced as fixing the classes that you must create yourself as part of "Part B"?

a) Class Structure Diagram b) Initial Class Diagram c) Entity Relationship Diagram d) Sequence Diagram

**Question 4: Memorization (ghi nhớ)**

Based on the assignment specification and the provided code, what mathematical calculation involving the real and imaginary parts of a DeathEater's complex mana is used to determine its base CombatScore?

a) The sum of the real and imaginary parts. b) The product of the real and imaginary parts. c) The magnitude (or modulus) of the complex number. d) The square root of the sum of the real and imaginary parts.

Okay, based on the provided sources and our conversation history, here are four quiz-style questions categorized as "Comprehension":

**Question 1: Comprehension (thông hiểu)**

Based on the provided code snippets, which show that public class Paladin extends Knight and public class Knight extends Fighter, what can you correctly infer about the inheritance relationship involving the Paladin class within this assignment's structure?

a) Paladin directly inherits from Fighter. b) Paladin is a subclass of Knight, and therefore also indirectly a subclass of Fighter. c) Knight is a subclass of Paladin. d) Paladin and Fighter are completely unrelated classes.

**Question 2: Comprehension (thông hiểu)**

The @Override annotation is used just before the getCombatScore method in the source code provided for classes like DeathEater, Knight, Paladin, and Warrior. Based on your understanding of the standard function of this Java annotation, what does its consistent presence before getCombatScore in these classes most likely imply about the method?

a) The method is intended to be abstract and must be implemented by any further subclasses.

b) The method is a static helper method used for calculations.

c) The method is providing a specific implementation for a method that is already declared in a superclass or an interface implemented by these classes (such as the Combatable interface).

d) The method is only accessible from within the same Java package.

**Question 3: Comprehension (thông hiểu)**

Examine the getCombatScore method implementations for both the Knight and Warrior classes. Both classes contain an if statement checking if(getWp() == 1.0), followed by the exact same logic: if (getBaseHp() > 999) { return 999; } return getBaseHp();. Given that both Knight and Warrior extend Fighter, what can you reasonably infer from this identical code block in these two subclasses?

a) This specific combat calculation rule (getBaseHp(), capped at 999) when WP == 1.0 is unique to Knight and Warrior and doesn't relate to their parent Fighter.

b) This pattern suggests that the combat rule for WP == 1.0 is a common behavior or requirement that might originate from their shared superclass (Fighter) or the Combatable interface, even if not explicitly shown in the source snippets for the parent classes.

c) The getWp() == 1.0 condition only affects the combat score if the Battle.GROUND value is also a square number or a prime number.

d) The Fighter class must be an interface that only defines the getWp() method.

**Question 4: Comprehension (thông hiểu)**

The DeathEater class source code shows that it extends Monster and its getCombatScore method calls getMana() on itself, and then calls getRe() and getIm() on the object returned by getMana(). Based on standard object-oriented programming principles regarding inheritance and method calls, what can you confidently infer must be true about the Monster class?

a) Monster must implement the Combatable interface.

b) Monster must be an interface.

c) Monster must define or inherit a method named getMana() that is accessible to its subclasses and returns an object which itself has getRe() and getIm() methods.

d) Monster must be a concrete class and cannot be abstract.