1. **Summary of Access Modifiers**

| **Modifier** | **Class** | **Same Package** | **Subclass** | **Different Package** |
| --- | --- | --- | --- | --- |
| **private** | Yes | No | No | No |
| **default** | Yes | Yes | No | No |
| **protected** | Yes | Yes | Yes | No (unless subclass) |
| **public** | Yes | Yes | Yes | Yes |
| 1. **Encapsulation** is one of the fundamental concepts of object-oriented programming (OOP). It refers to the bundling of data (fields) and methods (functions) that operate on the data into a single unit, typically a class. Encapsulation restricts direct access to some of the object's components, which is a way of preventing accidental interference and misuse of the data.   In Java, encapsulation is implemented by:   1. **Private Data Members**: Declaring the fields (data) of a class as private to prevent direct access from outside the class. 2. **Public Methods (Getters and Setters)**: Providing public methods (getters and setters) to access and update the value of private fields. This ensures controlled access to the class data. |  |  |  |  |

**Differences Between Interface and Abstract Class:**

| **Aspect** | **Interface** | **Abstract Class** |
| --- | --- | --- |
| **Methods** | Can only have abstract methods (until Java 8). After Java 8, can have default and static methods. | Can have both abstract and concrete methods. |
| **Fields** | Variables are public, static, and final by default. | Can have non-final fields. |
| **Multiple Inheritance** | A class can implement multiple interfaces (supports multiple inheritance). | A class can extend only one abstract class. |
| **Access Modifiers** | Methods are implicitly public and abstract; cannot have private or protected methods (until Java 9, which allows private methods). | Can have public, protected, or private methods. |
| **Constructors** | Cannot have constructors. | Can have constructors and can be used for initialization. |
| **Purpose** | Used to define a contract or behavior that implementing classes must follow. | Used to share code and force subclasses to implement certain methods. |
| **Speed** | Interface methods are slower due to dynamic method resolution. | Abstract class methods are faster because they are resolved at compile time. |

| **Aspect** | **throw** | **throws** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Purpose** | Used to explicitly throw an exception | Used in method declarations to indicate exceptions that might be thrown |

|  |  |  |
| --- | --- | --- |
| **Location** | Used inside a method or block of code | Used in the method signature |

|  |  |  |
| --- | --- | --- |
| **Usage** | Follows an instance of Throwable | Followed by one or more exception types |

|  |  |  |
| --- | --- | --- |
| **Checked vs. Unchecked** | Can throw both checked and unchecked exceptions | Typically used for c |