Comparison of Design patterns in Java and C++

Category	Pattern	Java	C++
Creational	Factory method	Uses abstract keyword to declare factory methods in factory classes to be later implemented by subclasses	Uses static pointers to declare factory methods
	Abstract factory	Uses ‘abstract’ keyword to make abstract factories classes and interfaces	Uses ‘class’ keyword and pointers to create abstract factories and the ‘new’ keyword to create concrete factories which is later used to create concrete objects
	Builder	Defines classes with creation methods for creating or building complex objects.	An abstract base class declares the standard construction process, and concrete derived classes define the appropriate implementation for each step of the process. Uses struct and class keyword in process.
	Singleton	Uses the public keyword to define a single point of access method for classes	Make the class responsible for its own global pointer and "initialization on first use" (by using a private static pointer and a public static accessor method)
	Prototype	Uses the cloneable interface for implementing class prototypes	A superclass defines a clone method and subclasses implement this method to return an instance of the class

Structural	Adapter	The adapter class uses the extend keyword to extend another class to make it compatible with another class	An abstract base class is created that specifies the desired interface. An "adapter" class is defined that publicly inherits the interface of the abstract class, and privately inherits the implementation of the legacy component. This adapter class "maps" or "impedance matches" the new interface to the old implementation.
	Bridge	abstraction and implementation to decouple classes into several related hierarchies	abstraction and implementation
	Composite	Uses inheritance to hierarchically implement an object tree	Uses inheritance and polymorphism to implement scalar/primitive classes and vector/container classes
	Decorator	Uses interface keyword to wrap subclasses and allow the subclasses to dynamically add new behaviours to objects	Uses the concept of wrapping-delegation which involves pointers to help add new behaviours to objects dynamically
Behavioural	Chain of responsibility	Defines an abstract class with series of methods including abstract methods for other classes to implement and handle a chain of actions independently	Uses pointers and classes and defines a chain method in the base class for delegating to the next object.
	Command	Applies the concept of inheritance and encapsulation to turn actions into objects	Applies the concept of inheritance and encapsulation to turn actions into objects
	Observer	Defines event listeners based on inheritance and encapsulation	Models the "independent" functionality with a "subject" abstraction and Models the "dependent"

			functionality with "observer" hierarchy
	Null object	Encapsulates the absence of an object by providing a substitutable alternative that offers suitable default do nothing behaviour	Similarly, provides a class that checks for null and returns a boolean
	Mediator	Uses ‘interface’ keyword to declare mediators which are later implemented by classes that intend to communicate with each other	Uses classes and pointers to implement mediators