perplexity

Difference between all the creational design pattern

Here is a concise comparison of the main **Creational Design Patterns** in Java, highlighting their purpose, usage, and key differences based on the search results and common understanding:

Pattern	Purpose	What it Creates	Key Characteristics & When to Use
Singleton	Ensures a class has only **one instance** and provides a global access point to it.	Single unique instance	Use when exactly one object is needed throughout the application, e.g., logging, configuration, thread pools.
Factory Method	Defines an interface for creating an object, but lets subclasses decide which class to instantiate.	One product per factory method call	Use when a class wants its subclasses to decide which object to create, promoting loose coupling.
Abstract Factory	Provides an interface for creating **families of related or dependent objects** without specifying concrete classes.	Multiple related products (families)	Use when you need to create families of related objects that should be used together, e.g., UI themes or platforms.
Builder	Separates the construction of a complex object from its representation, allowing step-by-step creation.	Complex objects with many optional parts	Use when objects have many parameters (especially optional) or complex construction logic, improving readability.
Prototype	Creates new objects by **cloning existing instances** instead of creating from scratch.	Copies of existing objects	Use when object creation is costly or complex, and you want to create new objects by copying existing ones.

Summary of Differences

- **Singleton** focuses on **single instance control**.
- **Factory Method** focuses on **deferring instantiation to subclasses**, creating one product.
- **Abstract Factory** creates **families of related products** without specifying concrete classes.
- **Builder** focuses on **stepwise construction** of complex objects, improving clarity and flexibility.
- **Prototype** uses **cloning** to create new objects, avoiding costly instantiation.

When to Use Which?

Scenario	Recommended Pattern
Need a single shared resource instance	Singleton
Need to create objects but want subclasses to decide the exact type	Factory Method
Need to create families of related objects	Abstract Factory
Need to build complex objects step-by-step	Builder
Need to create copies of existing objects	Prototype

Additional Notes

- Creational patterns **increase flexibility and reuse** by abstracting and controlling object creation.
- They help **decouple client code** from concrete implementations.
- Patterns like **Abstract Factory** and **Builder** solve problems that arise with complex object creation and family consistency.
- **Prototype** can improve performance when object creation is expensive.

This comparison is based on authoritative sources like Refactoring.Guru, DigitalOcean tutorials, and summarized Java design pattern guides[1] [2] [3]. If you want, I can provide detailed examples or code snippets for any specific pattern.

⁂

1. https://refactoring.guru/design-patterns/creational-patterns
2. https://www.kodnest.com/blog/15-essential-design-patterns-explained-with-java-examples
3. https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial