CREATIONAL DESIGN PATTERNS – CHEAT SHEET

---

1. Singleton

Purpose: One instance, global access.
Thread-safety rules:

- Without synchronization → multiple threads can create multiple instances.
- JVM **class initialization** is synchronized → static fields / static blocks are safe.
- Multiple ClassLoaders can break Singleton (1 per loader).

Variations:

Variation	Thread Safe?	Lazy?	Notes
Eager init (static final)	✅	❌	Simple, no exception handling
Static block	✅	❌	Same as eager, but can handle exceptions/setup
Lazy (synchronized method)	✅	✅	Easy, but sync cost on every call
Double-checked locking + volatile	✅	✅	Efficient lazy init
Bill Pugh (inner static holder)	✅	✅	Uses JVM init safety, no locks
Enum Singleton	✅	❌	Reflection & serialization proof

Key clarifications from today:

- **Static block safety** comes from JVM `<clinit>` lock — only one thread runs init, others wait.
- `new` itself does not guarantee single object → without guard, multiple threads create multiple objects.
- Bill Pugh works because **inner class loads once** and **class init is atomic**.

---

2. Builder

Purpose: Step-by-step construction, especially when object has optional parameters.
Immutable vs Mutable:

- **Immutable:** All fields final, no setters after build → thread-safe.
- **Mutable:** Fields can be changed after build → allows updates.

Variations:

Variation	Notes
Separate Builder class	Decouples from product, more boilerplate
Static inner Builder in Product (Joshua Bloch)	Most common, makes product immutable
Fluent builder	Chaining methods for readability
Director + Abstract Builder (GoF)	Builder builds parts, Director controls sequence
Step builder	Enforces order of setting fields via interfaces

Key clarifications from today:

- The **private constructor + static inner builder** is just one variation (Joshua Bloch style).
- Director version is useful when **build sequence is fixed or reused**.
- Step-by-step abstraction via Director is separate from product's optional field handling.

---

3. Factory Method

Purpose: Let subclasses decide which concrete object to create.
Misconception resolved today:

- In a car rental platform, "factory" doesn't mean building a *physical* car — it means creating a **software object** representing a car and adding it to inventory.

---

4. Abstract Factory

Purpose: Create **families** of related objects without specifying concrete classes.
Example: NYIngredientFactory vs ChicagoIngredientFactory in a pizza app.

---

5. Prototype

Purpose: Create new objects by copying an existing one (cloning).
Use when object creation cost is high and you need many similar objects.

---

6. Builder vs Factory

Builder	Factory
Step-by-step assembly	One-shot creation
Can handle many optional params	Fixed creation logic
Focuses on **how** to build	Focuses on **what** to build

---

7. Lazy vs Eager Initialization

- **Eager:** Create at class init time (static final / static block) → JVM thread-safe, but may waste memory if never used.
- **Lazy:** Create on first access → must handle thread safety manually, unless using Bill Pugh or similar.

JVM Class Loading & Initialization

- **Class Loaders:** Bootstrap → Platform → Application → Custom.
- **Initialization triggers:**
  - Access to static field (non-constant)
  - Static method call
  - `new` object creation
  - Reflection (`Class.forName("...")`, initialize=true)
- `<clinit>` is synchronized by JVM → only one thread runs it, others wait.
- **Safe publication:** After `<clinit>` completes, all threads see fully constructed static objects.

Key “Aha” moments from today

1. **Static block singleton** is safe because JVM synchronizes class initialization.
2. **Bill Pugh singleton** works by using inner-class loading + class init guarantee.
3. `new` without synchronization does not protect from multiple objects in multi-threaded code.
4. Static field assignment = eager init; lazy init in method needs explicit guard.
5. Factory in rental system = object creation, not real-world manufacturing.
6. Builder variations depend on where the Builder lives (inside product or separate) and whether a Director controls the sequence.
7. Immutable products in Builder are safer in multi-threaded contexts.
8. Director abstracts *sequence*, not object creation itself.