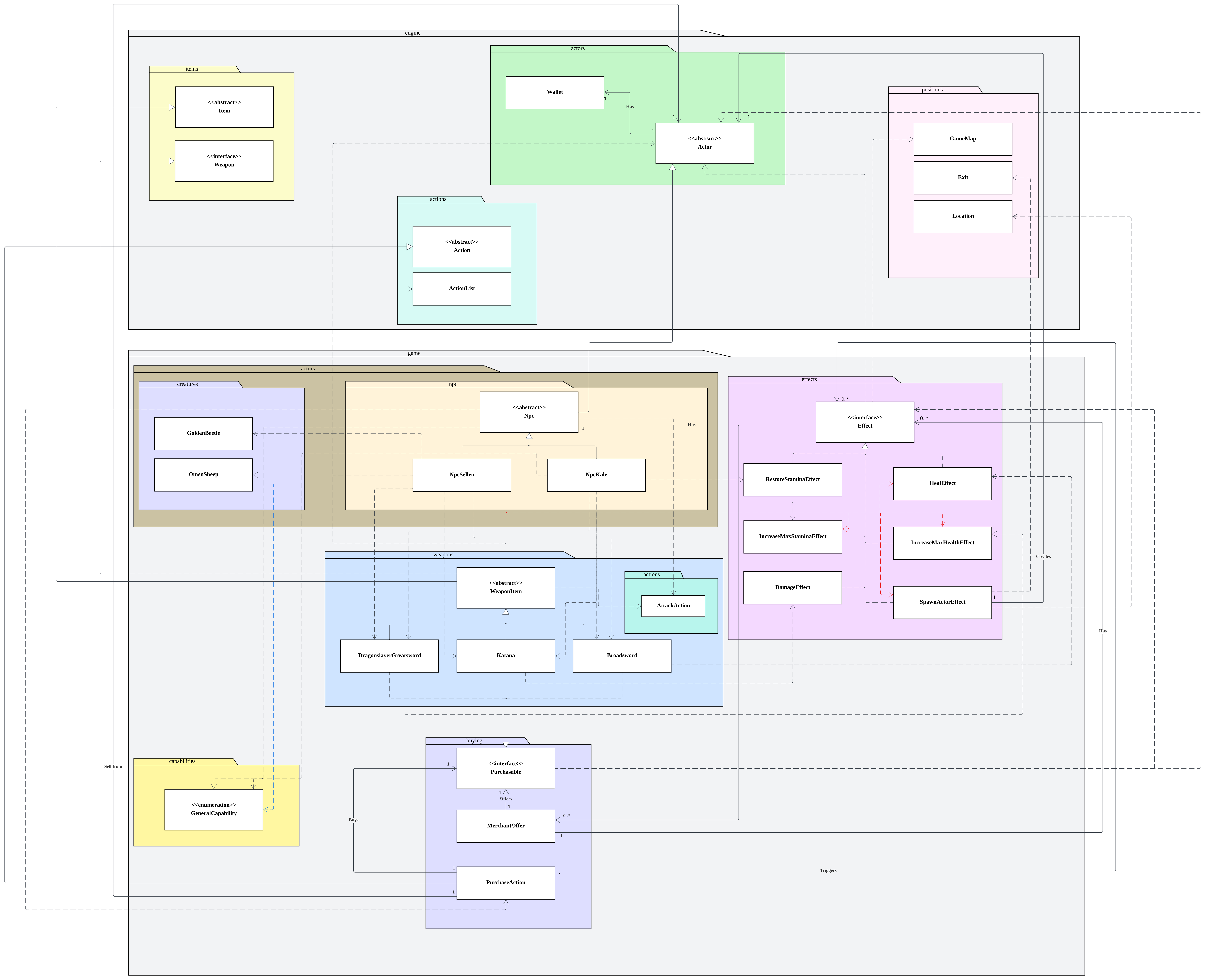
engine
items
<<abstract>> Item
<<interface>> Weapon
actors
Wallet
Has
<<abstract>> Actor
actions
<<abstract>> Action
ActionList
positions
GameMap
Exit
Location
game
actors
creatures
GoldenBeetle
OmenSheep
npc
<<abstract>> Npc
NpcSellen
NpcKale
effects
<<interface>> Effect
RestoreStaminaEffect
HealEffect
IncreaseMaxStaminaEffect
IncreaseMaxHealthEffect
DamageEffect
SpawnActorEffect
Creates
weapons
<<abstract>> WeaponItem
actions
AttackAction
DragonslayerGreatsword
Katana
Broadsword
buying
<<interface>> Purchasable
Offers
MerchantOffer
PurchaseAction
Buys
Triggers
Sell from
capabilities
<<enumeration>> GeneralCapability
Has
1
0..*

Design Rationale: Purchasing System & Weapon Implementation

1. Introduction

This rationale details the design for a system enabling Player weapon purchases from NPC merchants (REQ4). It covers the purchase process, layered base and merchant-specific effects, UI constraints (single-menu, one-tick actions), and future extensibility, prioritizing robust, maintainable, object-oriented design.

2. Design Goals

- Enable weapon purchases in the main menu, costing one tick.
- Support base and merchant-specific effects with extensibility for new merchants, items, and effects, requiring minimal system changes (OCP).

3. Design Analysis & Decisions

1. Core Purchase & Effect System

This feature addresses the overall architecture for managing item purchases, including varying costs and the application of multiple, layered effects.

Option A: Monolithic Logic in Player and PurchaseAction

- **Description:** All logic for identifying items, checking prices, deducting runes, transferring items, and applying all specific base and merchant-dependent effects would be hardcoded within a single large class, likely the Player class and PurchaseAction.
- **Maintainability:** Extremely poor. This class would become a super complex class, difficult to understand, modify, and test.

Pros	Cons
Simpler for a small, static set of items.	Violates SRP (complex Player logic) and OCP (modifications needed for new items).

Option B: **Interface-Driven Design**

- **Description**: Purchasable interface defines weapon base effects with methods like getting the base effects and transferring the item to buyer. MerchantOffer holds a Purchasable item, price, and merchant-specific Effect in an ArrayList. PurchaseAction will handle runes, transfer, and effect application.
- **Maintainability**: High. New items or effects are added without modifying core classes.

Pros	Cons
Adheres to SRP, OCP, and DRY, isolating components for testability.	More classes increase complexity

Decision & Justification

- **Chosen Design:** Option B.
- **Rationale:** Option B ensures modularity and extensibility (SRP, OCP). For instance, a new Fire Sword with a Burn effect can implement Purchasable and use a new MerchantOffer without changing PurchaseAction (OCP). If Kale changes his Broadsword offer to include a StrengthBuffEffect, only Kale's MerchantOffer instance for the Broadsword needs to be updated with this new Effect. The Broadsword class and the PurchaseAction remain untouched.

2. UI Integration & Action Cost

This feature ensures purchases are integrated into the main action menu and cost one game tick.

Option A: Npc Base Class Action Integration

- **Description:** The Npc base class's allowableActions() generates PurchaseAction instances for each MerchantOffer and adds to menu. These actions are processed if the NPC has CAN_SELL, and the item, price will be added to the list returned by the NPC's allowableActions method if the Player has CAN_BUY.
- **Maintainability:** High. Leverages the engine's action system for simplicity.

Pros	Cons
NPCs define actions, PurchaseAction executes. (SRP)	Long action menu lists for many items.

Option B: Centralized Purchase Manager

- **Description:** A single PurchaseManager checks for nearby merchants, hardcodes their offers, and generates PurchaseAction instances for the main menu in the game loop.
- **Maintainability:** Poor. PurchaseManager requires modifications for new merchants or items, complicating updates.

Pros	Cons
Centralizes buying logic for a small, static set of merchants.	Violates SRP (mixes action generation and buying logic) and OCP (new offers modify PurchaseManager).
	Violates OCP (new offers modify PurchaseManager).

Decision & Justification

- **Chosen Design:** Option A.
- **Rationale:** Option A integrates buying into the main menu via Npc.allowableActions(), ensuring UX consistency. A new TradeAction can extend allowableActions() without changes (OCP). Purchasing is a clear action in menu, ensuring the "no separate menu" rule is met.

3. **Weapon Subclass Creation and NPC Inventory Integration**

This feature defines weapon subclasses and their integration into NPC inventories.

Option A: Subclasses with MerchantOffer Integration

- **Description:** Subclasses (Broadsword, DragonslayerGreatsword, Katana) extend WeaponItem, defining attributes (damage, hit rate) and implementing Purchasable for base effects. NPC subclasses define MerchantOffers in constructors, linking weapons to inventories.
- **Maintainability:** High. New weapons and offers are added via subclasses.

Pros	**Cons**
Adheres to SRP and OCP where subclasses encapsulates the logic.	More subclasses increase complexity.
Reuses WeaponItem and MerchantOffer (DRY), supports LSP.	

Option B: Subclasses with Centralized MerchantInventory

- **Description:** Weapon subclasses extend WeaponItem, but the MerchantInventory class hardcodes NPC offers, managing weapon availability.
- **Maintainability:** Poor. Updates require modifying MerchantInventory.

Pros	**Cons**
Centralizes offer management.	Violates OCP and DRY, where a new offer will need modification, and similar offer structures repeated.

Decision & Justification

- **Chosen Design:** Option A.
- **Rationale:** Option A ensures modularity (OCP) by using weapon subclasses implementing Purchasable and NPC MerchantOffers for inventories. For

instance, IceAxe implementing Purchasable with a FreezeEffect, and having a new NPC sell it via a new MerchantOffer, requires no modifications to existing NPC code or central systems.

4. **Effect Implementation Details**

This feature focuses on the implementation of specific purchase effects like healing, stat changes, and actor spawning, utilizing the Effect interface and its concrete classes.

Option A: Effect Logic in PurchaseAction

- **Description:** The PurchaseAction.execute() method would contain large if/else blocks or switch statements to determine which item is bought and apply all associated base and merchant-specific effects by calling Player/map methods.
- **Maintainability**: Poor. Complex logic requires modifications for new effects.

Pros	**Cons**
Simpler for few, static effects.	Violates SRP & DRY. Becomes unmaintainable and error-prone as effects grow.

Option B: Effect Interface and Concrete Classes

- **Description**: Effect interface defines applyEffect(). Concrete classes (like HealEffect, SpawnActorEffect) handle specific effects. Purchasable items (in getBasePurchaseEffects()) and MerchantOffer objects (in their list of additional effects) hold lists of these Effect objects. The PurchaseAction iterates through these lists and applies the effect for each one.

- **Maintainability**: High. New effects are added without modifying existing code.

Pros	**Cons**
Flexible, compositional, and highly maintainable. (OCP & DRY)	Results in a larger number of small, focused classes.

Decision & Justification

- **Chosen Design:** Option B.
- **Rationale:** Option B uses the Effect interface for flexibility (OCP, SRP). A GrantsInvisibilityEffect can be added without altering PurchaseAction. If a new effect is needed, only a new Effect class is required, ensuring maintainability over Option A's rigid conditionals.

4. Conclusion

This design for the purchasing system, centered around the Purchasable interface for items, MerchantOffer to define specific deals, a cohesive PurchaseAction for transaction orchestration, and a flexible Effect system (Strategy Pattern) for outcome management, prioritizing SRP, OCP, and DRY. This modular system supports future addition of items and effects.