From a Java programming standpoint, these requirements outline the necessary classes, their attributes (fields), behaviors (methods), and the relationships between them (like inheritance, association, and composition).

## Character Requirements

These requirements define the actors in the game world. This points to a main Character abstract class with several concrete subclasses.
 * The game must be populated by characters.
 * Characters can be either player-controlled or AI-controlled. This suggests an inheritance structure, for example, PlayerCharacter and AICharacter classes that extend a base Character class.

* All characters must have a name, health points, strength points, and an amount of money. In Java, these would be attributes (fields) in the Character class, such as String name;, int healthPoints;, int strengthPoints;, and double money;.

 * A character can be at only one location at any given time. This defines a one-to-one association between a Character object and a Location object.

 * Characters must be able to acquire skills, with no limit on the number of skills they can have. This implies a one-to-many relationship, best implemented in Java with a List<Skill> or Set<Skill> attribute within the Character class.

* Characters must be able to move to an adjacent location during a game turn. This would be a method, like move(Location destination).

 * When a character's health reaches zero, it dies. This is a state change that would trigger other actions.

 * Upon death, a character drops all items and money it is carrying. This would be a behavior or method, onDeath(), that transfers the character's inventory to the location's inventory.

## Item & Inventory Requirements

These requirements detail the objects characters can own and use. This suggests an Item parent class with various specialized subclasses.

* A character can carry a maximum of two items directly. This could be implemented with a fixed-size array or a list with a size constraint in the Character class.

* Some items are weapons. This points to an inheritance relationship: a Weapon class would extend the Item class.

* Some items are containers (like a backpack), which can hold other items. A Container class would also extend Item and would contain a collection of other Item objects (e.g., List<Item>), representing a composition relationship. This allows a character to indirectly carry more than two items.

* Some items are consumable. A Consumable class would extend Item.

* Consuming an item can increase a character's health or strength, or grant temporary special properties. This indicates a consume(Character target) method in the Consumable class.

* Once an item is consumed, it is destroyed. The consume() method would need to handle the object's removal from the game.

## World and Location Requirements

These requirements define the game's environment. This suggests World and Location classes.

* The game world is made up of many locations. A World class would likely hold a collection of all Location objects.

* Each location is a hexagon with six sides, which dictates that each location can have up to six neighbors. This is crucial for implementing character movement.

* Each location must have a terrain type and a traversal cost (in health points). These would be attributes of the Location class. The terrain type could be implemented using an Enum in Java (TerrainType.FARMLAND, TerrainType.DESERT, etc.).

* Locations can contain items and money that characters can pick up or drop. This means the Location class needs its own inventory, such as a List<Item> and a double for money.

## Interaction Requirements

These requirements define how characters engage with each other. This could be handled by an Interaction interface with different implementing classes.
 * Characters at the same location can interact.
 * There are two types of interaction: combat and trading. This suggests two classes, Combat and Trading, that might implement a common Interaction interface.

 * Combat: Health points are lost based on strength, skills, weapons, and a random factor. A method like executeCombat(Character attacker, Character defender) would contain this logic.

 * Trading: Characters can offer sets of items they are willing to trade. One character can then offer money or items for a desired item. The other character can accept or reject the trade. If accepted, the items/money are exchanged. This requires a complex set of methods to manage offers and the exchange of goods.

## "Wizzo" Game-Specific Requirements

These are extensions to the general platform, illustrating how the core design can be specialized for a specific game.

* There are two main character types: magical and non-magical.

  * Magical characters include Wizards and Elves.

  * Non-magical characters include Humans and Dwarves.

    * This is a clear inheritance hierarchy: MagicalCharacter and NonMagicalCharacter would extend Character, and specific types like Wizard would extend MagicalCharacter.

* Some skills are magical skills, like casting a spell. This suggests a MagicalSkill class extending Skill.

* Only magical characters are allowed to have magical skills. This is a business rule that must be enforced in the code, for instance, by overriding an addSkill() method in the MagicalCharacter class.

* Specific terrain types are defined: farmland, desert, marsh, and water.

* Specific items are defined: wands, potions, swords, axes, bread, and backpacks. These would be concrete classes extending the appropriate base Item types (e.g., Sword extends Weapon).

* Potions give a special property (invisibility, invincibility, endurance) for a fixed duration when consumed.