```java
package edu.odu.cs.cs330.items.creation;

import java.util.Map;

/**
 * This class handles all Item creation and lookup logic.
 */
@SuppressWarnings({
    "PMD.ClassNamingConventions",
    "PMD.DoubleBraceInitialization",
    "PMD.OnlyOneReturn",
    "PMD.LawOfDemeter"
})
public final class ItemFactory {
    /**
     * ItemFactory is a collection of static functions. There is no reason to
     * instatiate an ItemFactory object.
     */
    private ItemFactory()
    {
        // do not allow ItemFactory to be instantiated.
    }

    /**
     * This Lookup table contains a listing of all known keywords and the Item
     * sub-classes to which the correspond.
     */
    private static final Map<String, ItemCreationStrategy> KNOWN_ITEMS = new HashMap<>() {{
        put("Armour", ArmourCreation.construct());
        put("Armor", ArmourCreation.construct());
        put("Tool", ToolCreation.construct());
        put("Food", ConsumableCreation.construct());
        put("Potion", ConsumableCreation.construct());
        put("Disposable", ConsumableCreation.construct());
    }};

    /**
     * Get an item creation strategy
     *
     * @param type the item to be created
     *
     * @return A creation strategy for an item of the specified type, or null
     *     if the type is unknown
     */
    public static ItemCreationStrategy create final String type)
    {
        if (isNotKnown(type)) {
            return null;
        }

        return KNOWN_ITEMS.get(type);
    }

    /**
     * Determine whether a given item is known.
     *
     * @param type the item for which to query
```

```
59      *
60      * @return true if the type can be created and false otherwise
61      */
62     public static boolean isKnown(final String type)
63     {
64         return KNOWN_ITEMS.containsKey(type);
65     }
66
67     /**
68      * Determine whether a given item is **not** known.
69      *
70      * @param type the item for which to query
71      *
72      * @return true if the type can be created and false otherwise
73      */
74     public static boolean isNotKnown(final String type)
75     {
76         return !KNOWN_ITEMS.containsKey(type);
77     }
78
79     /**
80      * Determine how many "tokens" are required to create a given item.
81      *
82      * @param type the item for which to query
83      *
84      * @return number of required tokens if the type is known and -1 otherwise
85      */
86     public static int getNumberOfRequiredValues(final String type)
87     {
88         if (isNotKnown(type)) {
89             return -1;
90         }
91
92         return KNOWN_ITEMS.get(type).requiredNumberOfValues();
93     }
94 }
95
96
97
98
```