Here is a detailed set of notes based on the provided document on **Reflection in Java**:

Reflection in Java

- **Definition**:
  - Reflection is a Java feature that enables an executing program to examine or "introspect" its own structure and manipulate its internal properties.
  - Example: A class can dynamically retrieve and display its member names.
  - Unique to Java compared to languages like Pascal, C, or C++, which lack this feature.
- **Uses**:
  - JavaBeans: Visual manipulation of software components using builder tools that leverage reflection.
  - Runtime Inspection: Inspect and manipulate classes, fields, methods, and constructors.
  - Applications:
    - Introspection
    - Testing and debugging
    - Creating flexible frameworks.

Setting Up to Use Reflection

1. **Obtaining a `Class` Object**:
   - A `Class` object represents classes and interfaces in a running program.
   - Methods to obtain a `Class` object:
     - **Using `Class.forName`**:

       ```
       Class c = Class.forName("java.lang.String");
       ```

     - **Using predefined fields for primitive types**:

       ```
       Class c = int.class;
       Class c = Integer.TYPE;
       ```

2. **Retrieve Class Details**:
   - Use methods like `getDeclaredMethods()` to list all methods declared in a class.
3. **Manipulate Using Reflection API**:
   - Example:

     ```
     Class c = Class.forName("java.lang.String");
     Method[] methods = c.getDeclaredMethods();
     System.out.println(methods[0].toString());
     ```

Key Classes in `java.lang.reflect`

1. **Array**: Allows dynamic creation and manipulation of arrays.
2. **Constructor**: Provides information about constructors.
3. **Field**: Provides information about fields.
4. **Method**: Provides information about methods.
5. **Modifier**: Gives details on access modifiers of classes and members.
6. **ReflectPermission**: Grants reflection permissions for private/protected members.

Methods in `java.lang.reflect`

1. `getName()` : Returns the name of the class.
2. `getSuperclass()` : Returns the superclass reference.
3. `getInterfaces()` : Returns an array of interfaces implemented by the class.
4. `getModifiers()` : Provides flags indicating modifiers for a class.
5. `getDeclaredMethod(String name)` : Creates an object for invoking a method.
6. `invoke()` : Invokes a method at runtime.
7. `getDeclaredField(String fieldName)` : Retrieves private fields.
8. `setAccessible(boolean flag)` : Access fields irrespective of their modifiers.
9. `newInstance()` : Creates new instances of a class:
   - Zero-argument constructor: `Class.newInstance()` .
   - Any constructor: `Constructor.newInstance()` .

Advantages of Reflection

1. **Extensibility**:
   - Load and manipulate user-defined classes dynamically by using their fully qualified names.
2. **Development Tools**:
   - Class browsers enumerate class members.
   - Visual tools assist developers by leveraging reflection-based type information.
3. **Testing and Debugging**:
   - Debuggers inspect private members.
   - Test tools systematically invoke APIs to ensure high code coverage.

Disadvantages of Reflection

1. **Performance Overhead**:
   - Dynamic type resolution hinders Java Virtual Machine optimizations.
2. **Security Restrictions**:
   - Reflection requires runtime permissions, which might be restricted in secured environments like applets.
3. **Breaks Abstractions**:
   - Reflective access to private members can cause side effects.
   - Such code may be non-portable and sensitive to platform upgrades.

These notes summarize the key aspects of Java Reflection, focusing on its functionality, setup, associated classes, methods, and pros/cons. Let me know if you'd like to expand or modify any section!