Java Hashmap:

A `HashMap` in Java is a data structure that implements the `Map` interface, allowing you to store key-value pairs. It is part of the Java Collections Framework and is based on a hash table data structure. Here are some advantages of using a `HashMap`:

1. **Fast Retrieval**: HashMap provides constant-time (O(1)) average time complexity for basic operations like put() and get(). This makes it efficient for retrieving values based on their keys.

2. **Dynamic Sizing**: HashMap can dynamically resize itself to accommodate more elements, so you don't need to specify its size in advance.

3. **Uniqueness of Keys**: Keys in a HashMap must be unique. It ensures that you can't have duplicate keys, which can be useful for maintaining a collection of unique items.

4. **Null Values**: HashMap allows one key to be associated with a null value, meaning you can have key-value pairs where the value is absent or undefined.

5. **Iterating Over Entries**: You can easily iterate over the key-value pairs using iterators or enhanced for loops.

Here are two code examples illustrating how to use a `HashMap` in Java:

Example 1: Basic Usage of HashMap

```java
import java.util.HashMap;
import java.util.Map;

public class HashMapExample1 {
    public static void main(String[] args) {
        // Creating a HashMap with Integer keys and String values
        Map<Integer, String> studentMap = new HashMap<>();

        // Adding key-value pairs to the HashMap
        studentMap.put(1, "Alice");
        studentMap.put(2, "Bob");
        studentMap.put(3, "Charlie");

        // Retrieving values based on keys
        String studentName = studentMap.get(2);
        System.out.println("Student with key 2: " + studentName);
```

```
        // Iterating over the HashMap entries
        for (Map.Entry<Integer, String> entry : studentMap.entrySet()) {
            System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
        }
    }
}
```

Example 2: Handling Collisions

```java
import java.util.HashMap;
import java.util.Map;

public class HashMapExample2 {
    public static void main(String[] args) {
        // Creating a HashMap with String keys and Integer values
        Map<String, Integer> ageMap = new HashMap<>();

        // Adding key-value pairs to the HashMap
        ageMap.put("Alice", 25);
        ageMap.put("Bob", 30);
        ageMap.put("Charlie", 25);

        // Retrieving values based on keys
        int aliceAge = ageMap.get("Alice");
        System.out.println("Alice's age: " + aliceAge);

        // Iterating over the HashMap entries
        for (Map.Entry<String, Integer> entry : ageMap.entrySet()) {
            System.out.println("Name: " + entry.getKey() + ", Age: " + entry.getValue());
        }
    }
}
```

In the second example, even though both Alice and Charlie have the same age, they can still be stored in the HashMap because their keys are different. This illustrates how HashMap handles collisions internally by using a linked list or a balanced tree structure (depending on the JDK version) to store multiple key-value pairs with the same hash code.