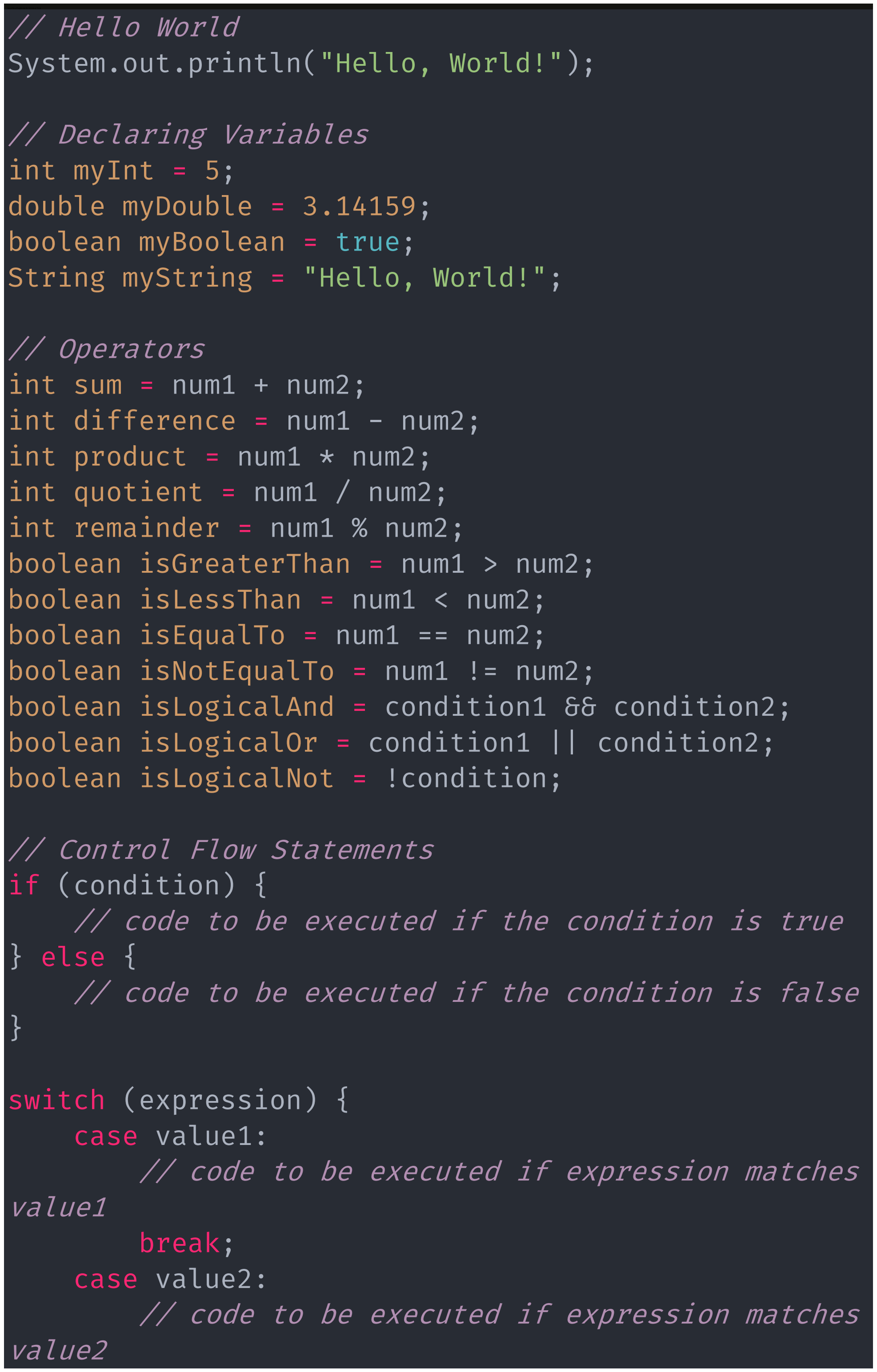
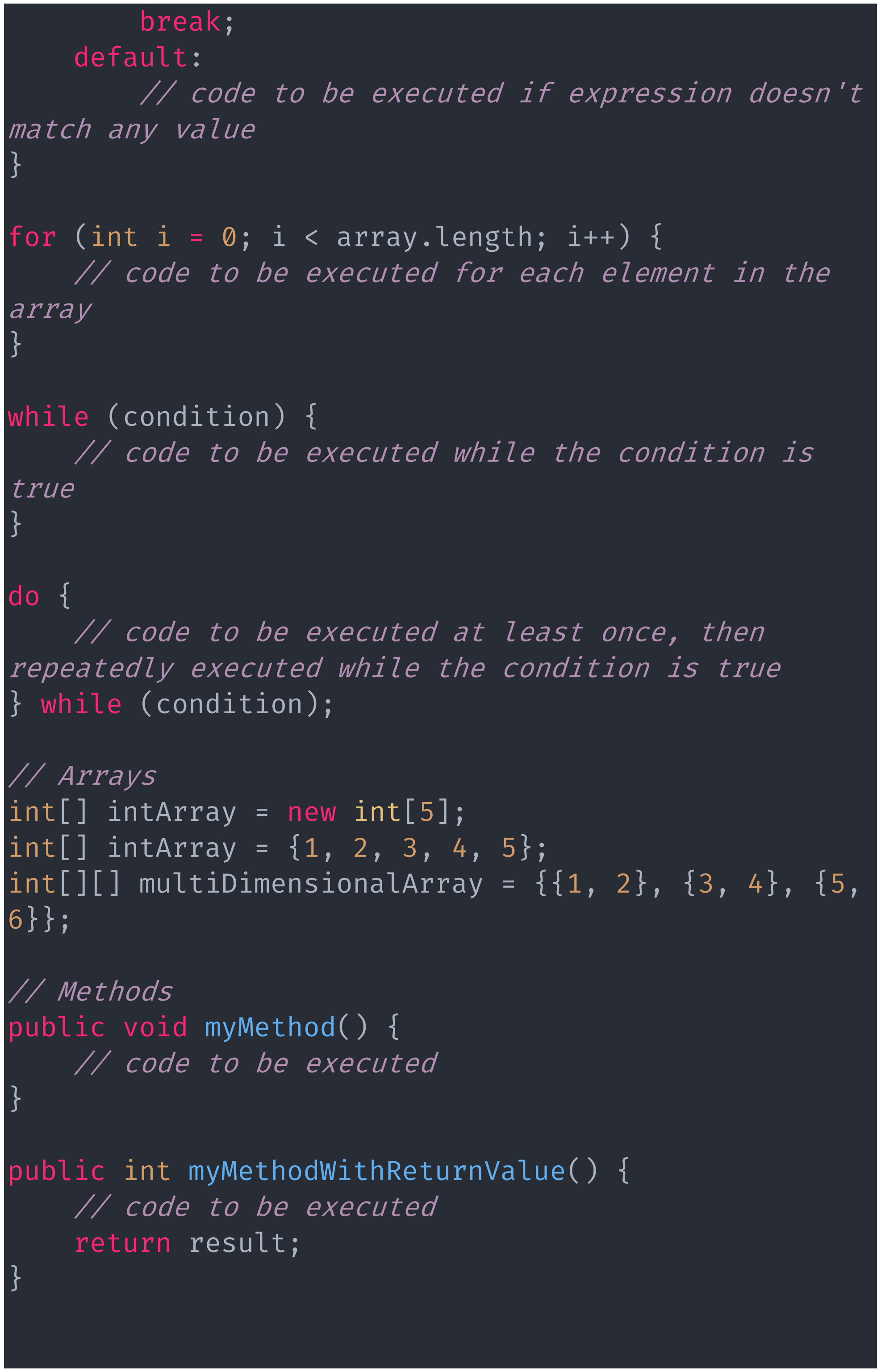


```java
// Hello World
System.out.println("Hello, World!");

// Declaring Variables
int myInt = 5;
double myDouble = 3.14159;
boolean myBoolean = true;
String myString = "Hello, World!";

// Operators
int sum = num1 + num2;
int difference = num1 - num2;
int product = num1 * num2;
int quotient = num1 / num2;
int remainder = num1 % num2;
boolean isGreaterThan = num1 > num2;
boolean isLessThan = num1 < num2;
boolean isEqualTo = num1 == num2;
boolean isNotEqualTo = num1 != num2;
boolean isLogicalAnd = condition1 && condition2;
boolean isLogicalOr = condition1 || condition2;
boolean isLogicalNot = !condition;

// Control Flow Statements
if (condition) {
    // code to be executed if the condition is true
} else {
    // code to be executed if the condition is false
}

switch (expression) {
    case value1:
        // code to be executed if expression matches
value1
        break;
    case value2:
        // code to be executed if expression matches
value2
```



```java
        break;
    default:
        // code to be executed if expression doesn't
match any value
}

for (int i = 0; i < array.length; i++) {
    // code to be executed for each element in the
array
}

while (condition) {
    // code to be executed while the condition is
true
}

do {
    // code to be executed at least once, then
repeatedly executed while the condition is true
} while (condition);

// Arrays
int[] intArray = new int[5];
int[] intArray = {1, 2, 3, 4, 5};
int[][] multiDimensionalArray = {{1, 2}, {3, 4}, {5,
6}};

// Methods
public void myMethod() {
    // code to be executed
}

public int myMethodWithReturnValue() {
    // code to be executed
    return result;
}
```

```
// Classes and Objects
public class MyClass {
    int myInt;
    double myDouble;
    boolean myBoolean;
    String myString;

    public MyClass(int myInt, double myDouble,
boolean myBoolean, String myString) {
        this.myInt = myInt;
        this.myDouble = myDouble;
        this.myBoolean = myBoolean;
        this.myString = myString;
    }

    public void myMethod() {
        // code to be executed
    }
}

MyClass myObject = new MyClass(5, 3.14159, true,
"Hello, World!");
myObject.myMethod();

// Exception Handling
try {
    // code that might throw an exception
} catch (ExceptionType1 e) {
    // code to handle exception of type
ExceptionType1
} catch (ExceptionType2 e) {
    // code to handle exception of type
ExceptionType2
} finally {
    // code to be executed regardless of whether an
exception was thrown
}
```

```
// Input and Output
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);
System.out.println("Enter a number:");
int num = scanner.nextInt();
System.out.println("You entered " + num);
```

Cheat sheet of strings

```java
// Creating Strings
String str1 = "Hello";
String str2 = new String("World");

// String Length
int length = str1.length();

// Concatenation
String concatenated = str1 + " " + str2;
String concatenated2 = str1.concat("
").concat(str2);

// Substring
String substring = str1.substring(1, 4);

// Character Access
char firstChar = str1.charAt(0);
char lastChar = str1.charAt(str1.length() - 1);

// Comparison
boolean isEqual = str1.equals(str2);
boolean isEqualIgnoreCase =
str1.equalsIgnoreCase(str2);

// Searching
int indexOfChar = str1.indexOf('l');
int indexOfString = str1.indexOf("lo");

// Replacing
String replaced = str1.replace('l', 'L');
String replacedString = str1.replace("llo",
"LLO");

// Splitting
String[] parts = str1.split(" ");
String[] partsLimit = str1.split(" ", 2);


// Conversion to Other Types
int intValue = Integer.parseInt("123");
double doubleValue = Double.parseDouble("3.14");
String strValue = String.valueOf(123);

// String Formatting
String formatted = String.format("Name: %s, Age:
%d", name, age);

// String Comparison
int compareResult = str1.compareTo(str2);
int compareIgnoreCaseResult =
str1.compareToIgnoreCase(str2);

// String Case Conversion
String lowercase = str1.toLowerCase();
String uppercase = str1.toUpperCase();

// Trimming
String trimmed = str1.trim();

// Checking Empty or Null
boolean isEmpty = str1.isEmpty();
boolean isNull = str1 == null;

// StringBuilder (mutable string)
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Hello");
stringBuilder.append(" World");
String result = stringBuilder.toString();
```

String buffer Cheatsheet

```java
// Creating a StringBuffer
StringBuffer buffer = new StringBuffer();

// Creating a StringBuffer with initial value
StringBuffer buffer = new StringBuffer("Hello");

// Appending
buffer.append(" World");

// Inserting
buffer.insert(5, ", Java");

// Deleting
buffer.delete(5, 11);

// Reversing
buffer.reverse();

// Updating a character at a specific index
buffer.setCharAt(0, 'h');

// Getting the length of the StringBuffer
int length = buffer.length();

// Getting the capacity of the StringBuffer
int capacity = buffer.capacity();

// Setting the capacity of the StringBuffer
buffer.ensureCapacity(20);

// Converting the StringBuffer to a String
String str = buffer.toString();
```

String builder cheatsheet

```java
// Creating a StringBuilder
StringBuilder sb = new StringBuilder();

// Creating a StringBuilder with initial capacity
StringBuilder sb = new StringBuilder(capacity);

// Appending content to StringBuilder
sb.append(str); // Appends a string
sb.append(obj); // Appends an object
sb.append(charArray); // Appends a character array
sb.append(charArray, offset, length); // Appends a portion of a character array
sb.append(booleanValue); // Appends a boolean value
sb.append(charValue); // Appends a character value
sb.append(intValue); // Appends an integer value
sb.append(longValue); // Appends a long value
sb.append(floatValue); // Appends a float value
sb.append(doubleValue); // Appends a double value

// Inserting content at a specific position
sb.insert(index, str); // Inserts a string at the specified index
sb.insert(index, obj); // Inserts an object at the specified index
sb.insert(index, charArray); // Inserts a character array at the specified index
sb.insert(index, charArray, offset, length); // Inserts a portion of a character array at the specified index
sb.insert(index, booleanValue); // Inserts a boolean value at the specified index
sb.insert(index, charValue); // Inserts a character value at the specified index
sb.insert(index, intValue); // Inserts an integer value at the specified index
sb.insert(index, longValue); // Inserts a long value at the specified index
sb.insert(index, floatValue); // Inserts a float value at the specified index
```

```java
sb.insert(index, doubleValue); // Inserts a double value at the specified index

// Deleting content from StringBuilder
sb.delete(startIndex, endIndex); // Deletes a portion of StringBuilder
sb.deleteCharAt(index); // Deletes a character at the specified index

// Reversing the contents of StringBuilder
sb.reverse(); // Reverses the content

// Retrieving the length of StringBuilder
int length = sb.length(); // Returns the number of characters in StringBuilder

// Converting StringBuilder to String
String str = sb.toString(); // Converts StringBuilder to String

// Modifying characters at specific positions
sb.setCharAt(index, newChar); // Sets the character at the specified index to a new character

// Modifying the capacity of StringBuilder
sb.ensureCapacity(capacity); // Ensures that the capacity of StringBuilder is at least equal to the specified capacity
sb.trimToSize(); // Trims the capacity of StringBuilder to the current length

// Checking if StringBuilder is empty
boolean isEmpty = sb.length() == 0; // Checks if StringBuilder has no characters

// Appending a formatted string
sb.append(String.format(format, args)); // Appends a formatted string using format specifiers
```

Arrays cheatsheet

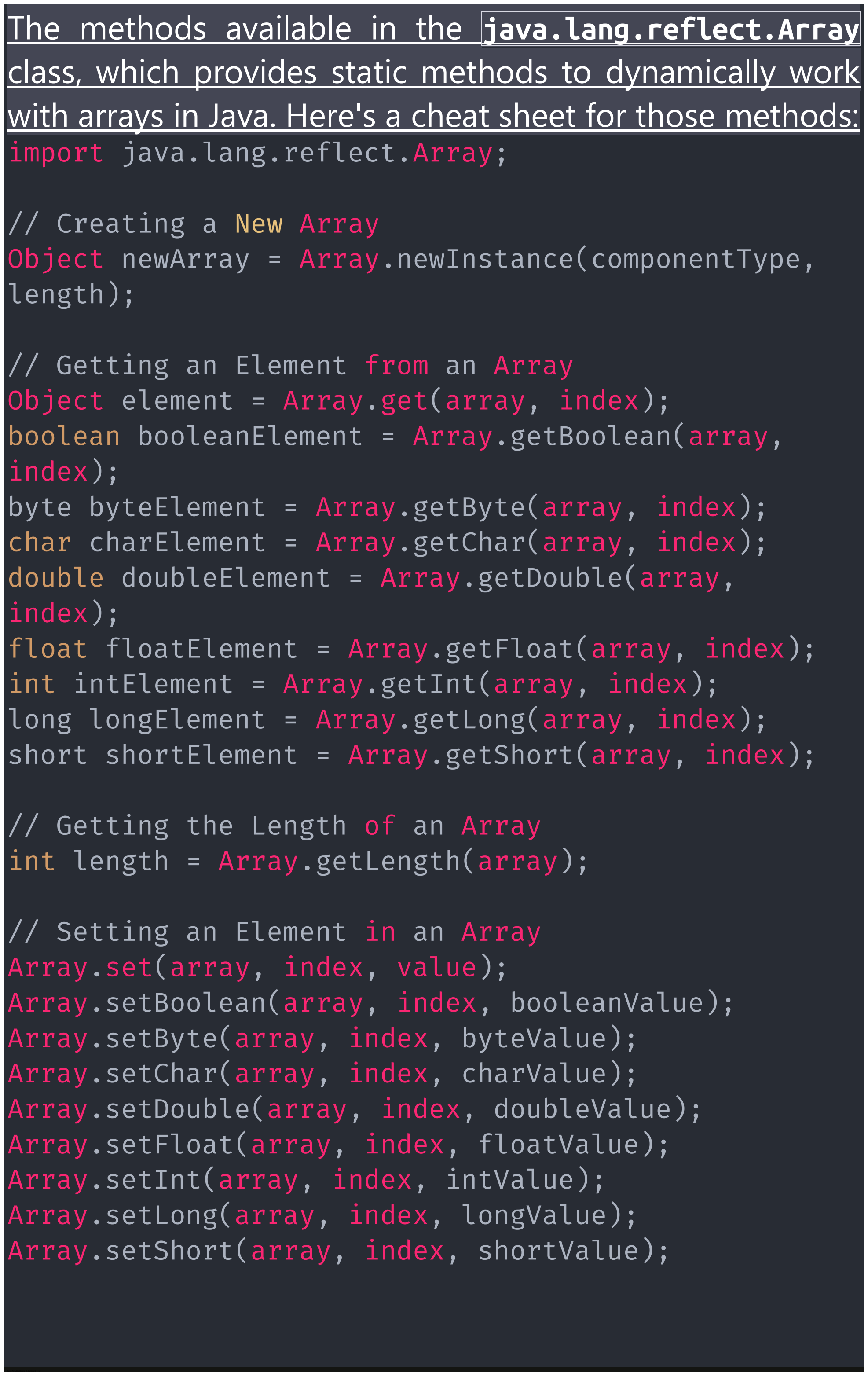
```java
import java.util.Arrays;

// Sorting Arrays
Arrays.sort(array); // Sorts the array in ascending
order

// Searching Arrays
int index = Arrays.binarySearch(array, key); //
Searches for the key in the sorted array

// Filling Arrays
Arrays.fill(array, value); // Fills the entire array
with the specified value

// Copying Arrays
DataType[] newArray = Arrays.copyOf(array, length);
// Creates a new array with the specified length
DataType[] newArray = Arrays.copyOfRange(array,
startIndex, endIndex); // Creates a new array with
elements from the specified range

// Comparing Arrays
boolean areEqual = Arrays.equals(array1, array2); //
Checks if two arrays are equal (element-by-element
comparison)

// Converting Arrays to Strings
String arrayString = Arrays.toString(array); //
Returns a string representation of the array

// Sorting Array in Descending Order (with a custom
comparator)
Arrays.sort(array, Collections.reverseOrder()); //
Requires import java.util.Collections
```

```java
// Checking if an Array Contains a Specific Value
boolean containsValue =
Arrays.asList(array).contains(value);

// Converting Array to List
List<DataType> list = Arrays.asList(array); //
Returns a fixed-size list backed by the array

// Checking if Two Arrays Overlap
boolean overlap = Arrays.overlaps(array1, array2);

// Checking if an Array is Empty
boolean isEmpty = array.length == 0;

// Checking Array Equality (element-by-element
comparison)
boolean areEqual = Arrays.deepEquals(array1,
array2); // For multidimensional arrays

// Hashing an Array
int hashCode = Arrays.hashCode(array);

// Sorting Multidimensional Arrays
Arrays.sort(array, Comparator.comparingInt(row ->
row[columnIndex]));

// Converting Array to Stream
Stream<DataType> stream = Arrays.stream(array);
```

Arraylist cheatsheet

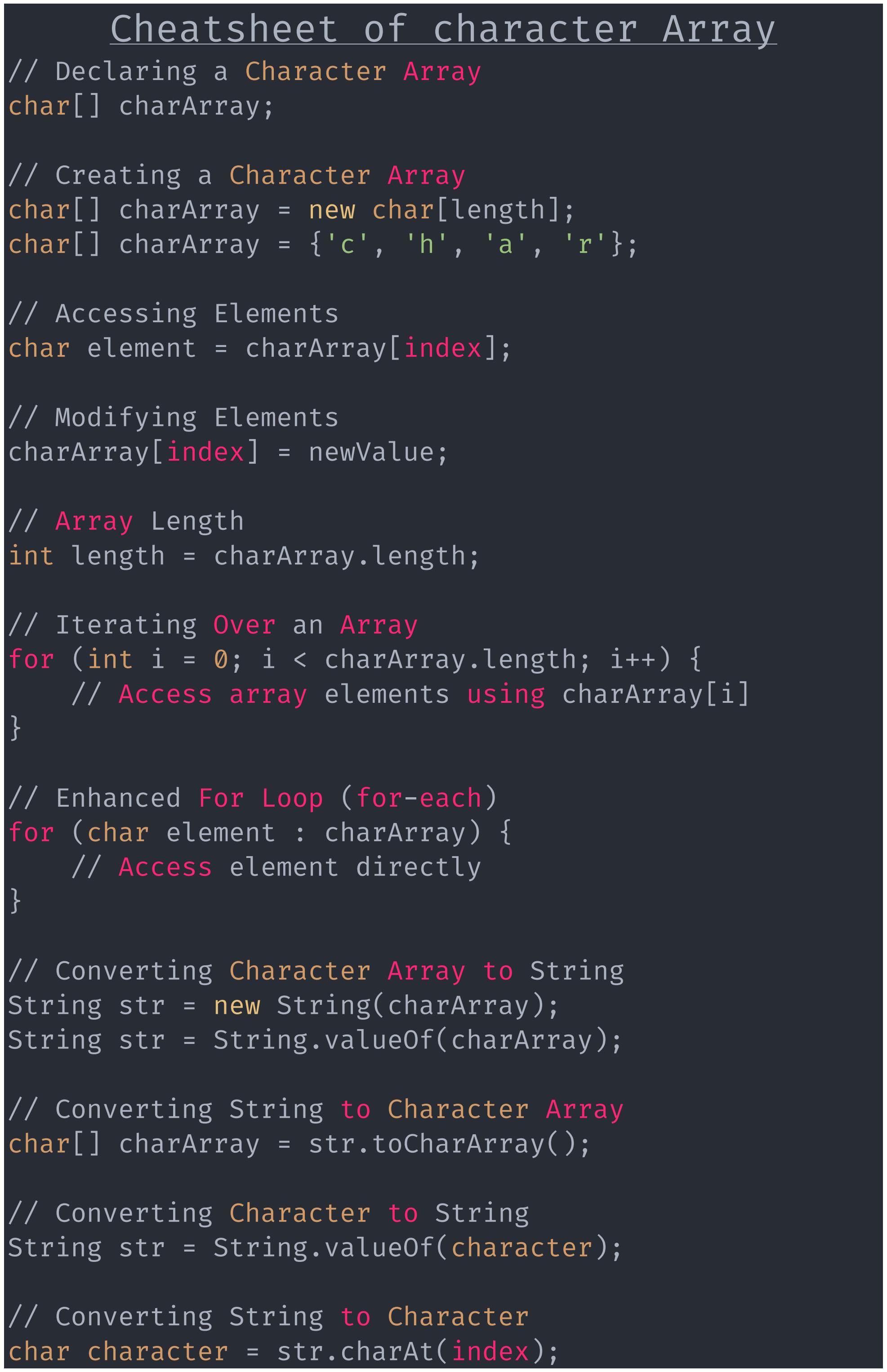
```java
import java.util.ArrayList;

// Creating an ArrayList
ArrayList<Type> list = new ArrayList<>();

// Adding elements to ArrayList
list.add(element); // Adds an element to the end of the list
list.add(index, element); // Inserts an element at the specified index

// Accessing elements in ArrayList
Type element = list.get(index); // Retrieves the element at the specified index

// Updating elements in ArrayList
list.set(index, newElement); // Replaces the element at the specified index with a new element

// Removing elements from ArrayList
list.remove(index); // Removes the element at the specified index
list.remove(element); // Removes the first occurrence of the specified element
list.clear(); // Removes all elements from the list

// Checking if ArrayList contains an element
boolean containsElement = list.contains(element); // Checks if the list contains the specified element

// Checking if ArrayList is empty
boolean isEmpty = list.isEmpty(); // Checks if the list is empty
```

```java
// Getting the size of ArrayList
int size = list.size(); // Returns the number of elements in the list


// Converting ArrayList to Array
Type[] array = list.toArray(new Type[0]); // Converts the list to an array of the specified type

// Iterating over ArrayList
for (Type element : list) {
    // Do something with the element
}

// Sorting ArrayList
Collections.sort(list); // Sorts the elements in the list (requires import java.util.Collections)

// Reversing the order of elements in ArrayList
Collections.reverse(list); // Reverses the order of elements in the list (requires import java.util.Collections)
```



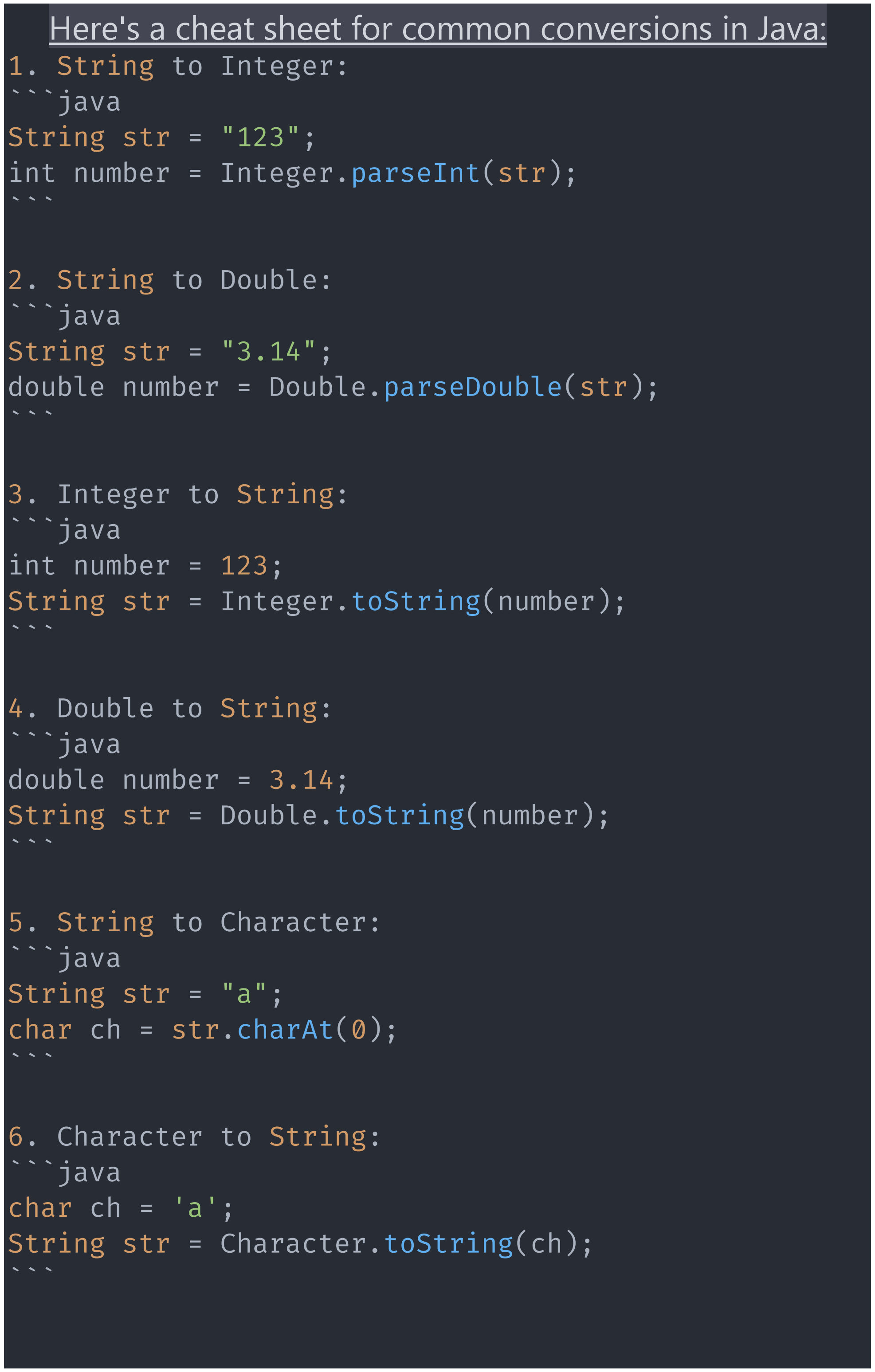
The methods available in the `java.lang.reflect.Array` class, which provides static methods to dynamically work with arrays in Java. Here's a cheat sheet for those methods:

```java
import java.lang.reflect.Array;

// Creating a New Array
Object newArray = Array.newInstance(componentType,
length);

// Getting an Element from an Array
Object element = Array.get(array, index);
boolean booleanElement = Array.getBoolean(array,
index);
byte byteElement = Array.getByte(array, index);
char charElement = Array.getChar(array, index);
double doubleElement = Array.getDouble(array,
index);
float floatElement = Array.getFloat(array, index);
int intElement = Array.getInt(array, index);
long longElement = Array.getLong(array, index);
short shortElement = Array.getShort(array, index);

// Getting the Length of an Array
int length = Array.getLength(array);

// Setting an Element in an Array
Array.set(array, index, value);
Array.setBoolean(array, index, booleanValue);
Array.setByte(array, index, byteValue);
Array.setChar(array, index, charValue);
Array.setDouble(array, index, doubleValue);
Array.setFloat(array, index, floatValue);
Array.setInt(array, index, intValue);
Array.setLong(array, index, longValue);
Array.setShort(array, index, shortValue);
```



Cheatsheet of character Array

```java
// Declaring a Character Array
char[] charArray;

// Creating a Character Array
char[] charArray = new char[length];
char[] charArray = {'c', 'h', 'a', 'r'};

// Accessing Elements
char element = charArray[index];

// Modifying Elements
charArray[index] = newValue;

// Array Length
int length = charArray.length;

// Iterating Over an Array
for (int i = 0; i < charArray.length; i++) {
    // Access array elements using charArray[i]
}

// Enhanced For Loop (for-each)
for (char element : charArray) {
    // Access element directly
}

// Converting Character Array to String
String str = new String(charArray);
String str = String.valueOf(charArray);

// Converting String to Character Array
char[] charArray = str.toCharArray();

// Converting Character to String
String str = String.valueOf(character);

// Converting String to Character
char character = str.charAt(index);
```


Here's a cheat sheet for common conversions in Java:

1. String to Integer:
```java
String str = "123";
int number = Integer.parseInt(str);
```

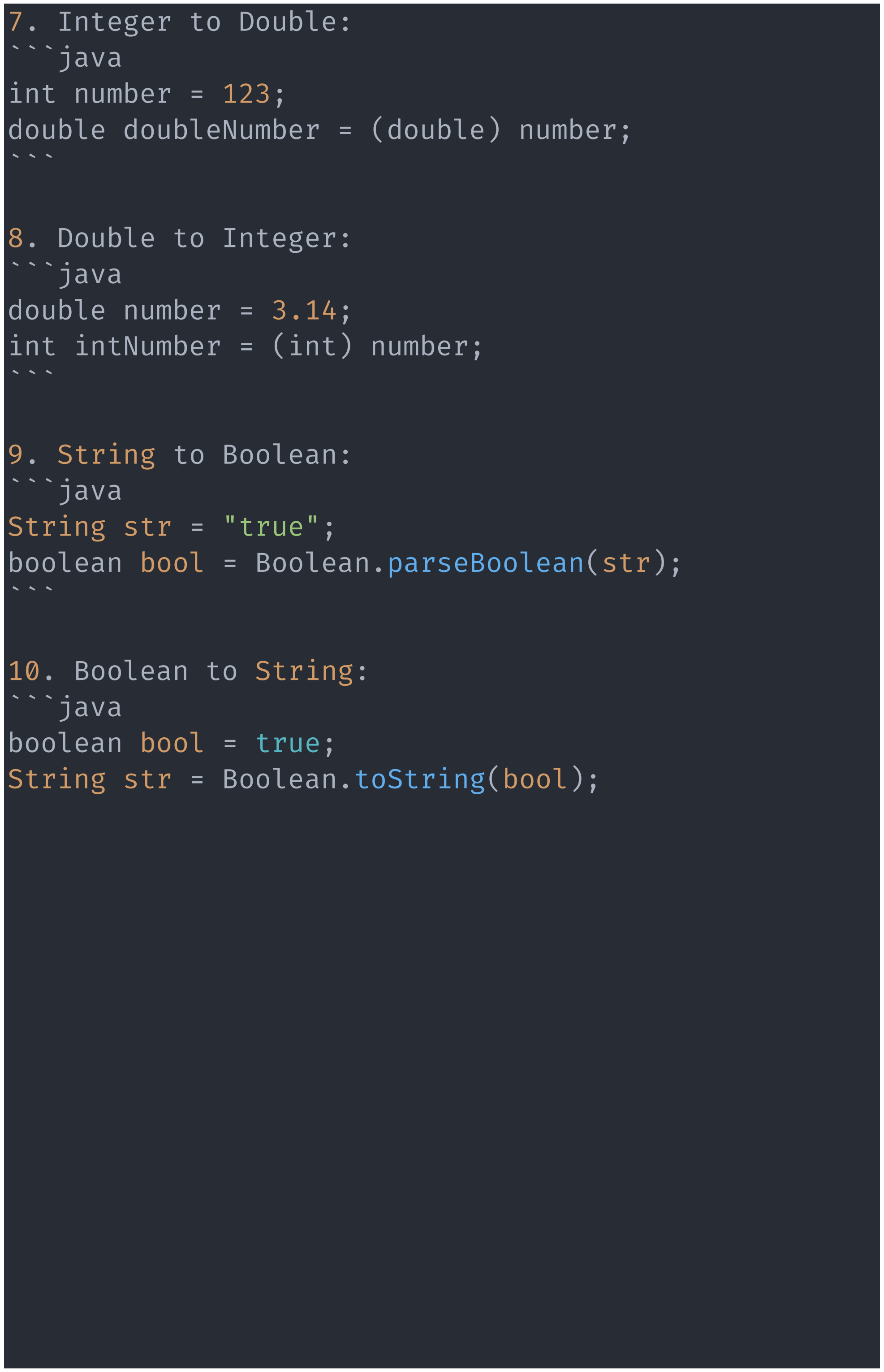
2. String to Double:
```java
String str = "3.14";
double number = Double.parseDouble(str);
```

3. Integer to String:
```java
int number = 123;
String str = Integer.toString(number);
```

4. Double to String:
```java
double number = 3.14;
String str = Double.toString(number);
```

5. String to Character:
```java
String str = "a";
char ch = str.charAt(0);
```

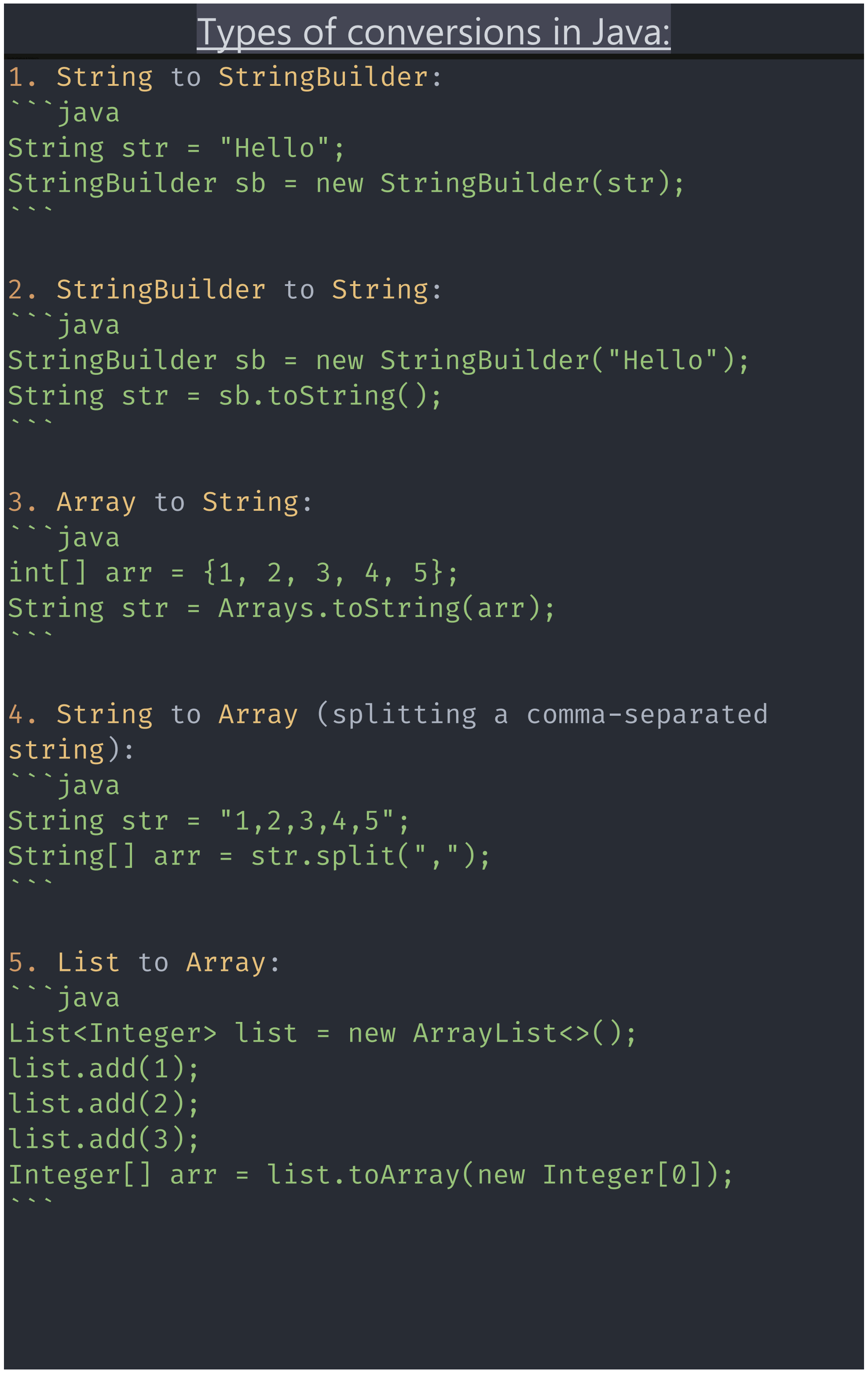
6. Character to String:
```java
char ch = 'a';
String str = Character.toString(ch);
```


7. Integer to Double:
```java
int number = 123;
double doubleNumber = (double) number;
```

8. Double to Integer:
```java
double number = 3.14;
int intNumber = (int) number;
```

9. String to Boolean:
```java
String str = "true";
boolean bool = Boolean.parseBoolean(str);
```

10. Boolean to String:
```java
boolean bool = true;
String str = Boolean.toString(bool);
```



Types of conversions in Java:

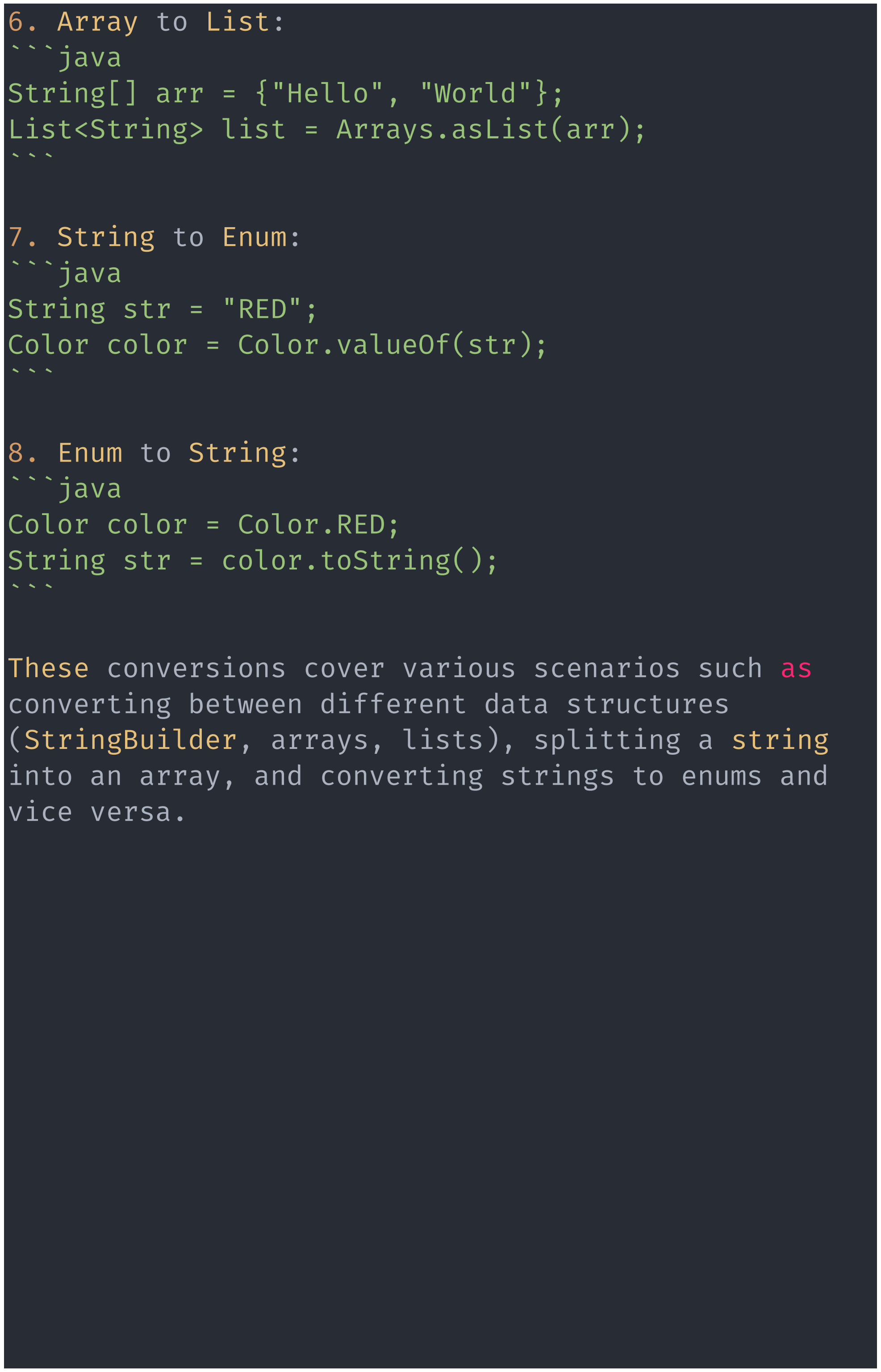
1. String to StringBuilder:
```java
String str = "Hello";
StringBuilder sb = new StringBuilder(str);
```

2. StringBuilder to String:
```java
StringBuilder sb = new StringBuilder("Hello");
String str = sb.toString();
```

3. Array to String:
```java
int[] arr = {1, 2, 3, 4, 5};
String str = Arrays.toString(arr);
```

4. String to Array (splitting a comma-separated string):
```java
String str = "1,2,3,4,5";
String[] arr = str.split(",");
```

5. List to Array:
```java
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
Integer[] arr = list.toArray(new Integer[0]);
```



6. Array to List:
```java
String[] arr = {"Hello", "World"};
List<String> list = Arrays.asList(arr);
```

7. String to Enum:
```java
String str = "RED";
Color color = Color.valueOf(str);
```

8. Enum to String:
```java
Color color = Color.RED;
String str = color.toString();
```

These conversions cover various scenarios such as converting between different data structures (StringBuilder, arrays, lists), splitting a string into an array, and converting strings to enums and vice versa.

cheat sheet for commonly used packages and functions in Java:

1. Math Package:
```java
import java.lang.Math;

Math.abs(-10); // Absolute value: 10
Math.sqrt(25); // Square root: 5.0
Math.pow(2, 3); // Power: 8.0
Math.max(10, 20); // Maximum value: 20
Math.min(10, 20); // Minimum value: 10
Math.random(); // Random value between 0.0 and 1.0
```

2. String Package:
```java
import java.lang.String;

String str = "Hello World";
str.length(); // Length of the string: 11
str.charAt(0); // Character at index 0: 'H'
str.substring(6); // Substring from index 6: "World"
str.toUpperCase(); // Convert to uppercase: "HELLO WORLD"
str.toLowerCase(); // Convert to lowercase: "hello world"
str.indexOf('o'); // Index of first occurrence of 'o': 4
str.endsWith("ld"); // Check if ends with "ld": true
str.replace('o', 'a'); // Replace 'o' with 'a': "Hella
Warld"
```

3. Arrays Package:
```java
import java.util.Arrays;

int[] numbers = {5, 2, 8, 1, 4};
Arrays.sort(numbers); // Sort the array: {1, 2, 4, 5, 8}
Arrays.toString(numbers);
// Convert array to string: "[1, 2, 4, 5, 8]"
Arrays.binarySearch(numbers, 4);
// Binary search for value 4: 2
Arrays.copyOf(numbers, 3);
// Copy first 3 elements: {1, 2, 4}
Arrays.fill(numbers, 0);
 // Fill array with value 0: {0, 0, 0, 0, 0}
```

4. Prime Number Check:
```java
public boolean isPrime(int number) {
    if (number <= 1) {
        return false;
    }
    for (int i = 2; i <= Math.sqrt(number); i++) {
        if (number % i == 0) {
            return false;
        }
    }
    return true;
}
```

5. Date and Time Package:
```java
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

LocalDate date = LocalDate.now(); // Current date
LocalTime time = LocalTime.now(); // Current time
LocalDateTime dateTime = LocalDateTime.now();
                                 // Currentdate and time

DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
String formattedDateTime = dateTime.format(formatter); //
Format date and time

// Parsing a date or time string
LocalDate parsedDate = LocalDate.parse("2023-05-18");
LocalTime parsedTime = LocalTime.parse("10:30:00");
```

`SimpleDateFormat` Cheat Sheet:

1. Format Date to String:

```
Date date = new Date();
SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd");
String dateString = format.format(date);
```

2. Parse String to Date:

```
String dateString = "2023-05-18";
SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd");
Date date = format.parse(dateString);
```

3. Format Date and Time:

```
Date date = new Date();
SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String dateTimeString = format.format(date);
```

4. Parse Date and Time:

```
String dateTimeString = "2023-05-18 10:30:00";
SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
Date dateTime = format.parse(dateTimeString);
```

5. Formatting Options:
- `yyyy`: Year in four digits (e.g., 2023)
- `MM`: Month in two digits (e.g., 05)
- `dd`: Day in two digits (e.g., 18)
- `HH`: Hour in two digits (e.g., 10)
- `mm`: Minute in two digits (e.g., 30)
- `ss`: Second in two digits (e.g., 00)

`GregorianCalendar` Cheat Sheet:

1. Create a `GregorianCalendar` instance:
```java
GregorianCalendar calendar = new
GregorianCalendar();
```

2. Set specific year, month, and day:

```java
GregorianCalendar calendar = new
GregorianCalendar(2023, Calendar.MAY, 18);
```

3. Get the current year, month, and day:

```java
int year = calendar.get(Calendar.YEAR);
int month = calendar.get(Calendar.MONTH);
int day = calendar.get(Calendar.DAY_OF_MONTH);
```

4. Set specific hour, minute, and second:

```java
calendar.set(Calendar.HOUR_OF_DAY, 10);
calendar.set(Calendar.MINUTE, 30);
calendar.set(Calendar.SECOND, 0);
```

5. Get the current hour, minute, and second:
```java
int hour = calendar.get(Calendar.HOUR_OF_DAY);
int minute = calendar.get(Calendar.MINUTE);
int second = calendar.get(Calendar.SECOND);
```

6. Add or subtract time units:

```java
calendar.add(Calendar.DAY_OF_MONTH, 1); // Add one
day
calendar.add(Calendar.HOUR_OF_DAY, -2); // Subtract
two hours
```

7. Format `GregorianCalendar` to `Date`:

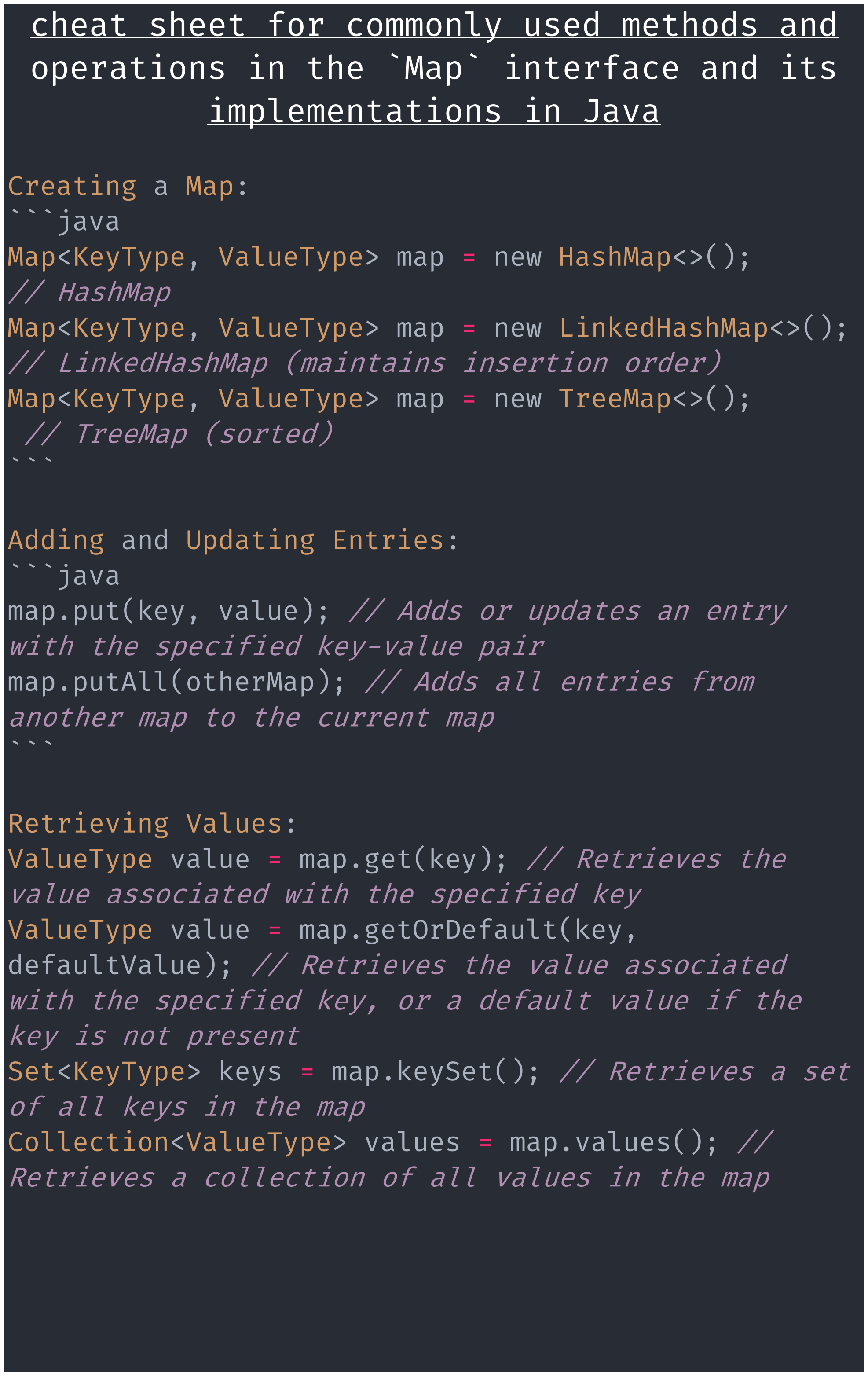
```java
Date date = calendar.getTime();
```

These are some basic operations with `SimpleDateFormat` and `GregorianCalendar` in Java. It's important to note that the `java.util.Date` and `java.util.Calendar` classes are considered outdated, and it is recommended to use the newer `java.time` classes introduced in Java 8 for date and time operations.

Set AND Hashset cheatsheet

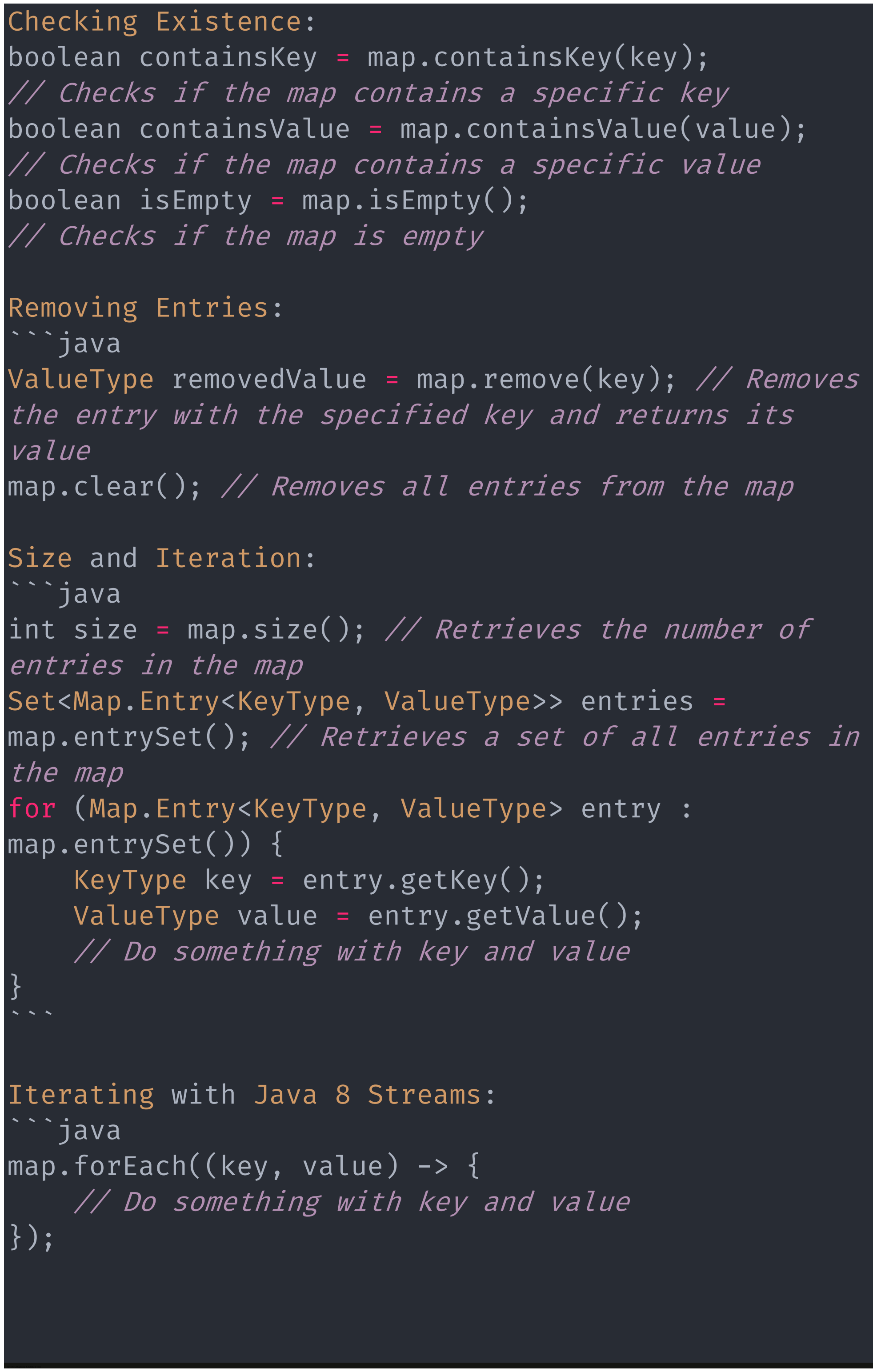
```java
// Importing the Set interface and HashSet class
import java.util.Set;
import java.util.HashSet;

// Creating a Set
Set<ElementType> set = new HashSet<>();

// Adding Elements
set.add(element1);
set.add(element2);

// Removing an Element
set.remove(element);

// Checking if an Element Exists
boolean containsElement = set.contains(element);

// Checking if the Set is Empty
boolean isEmpty = set.isEmpty();

// Getting the Size of the Set
int size = set.size();

// Iterating over the Set
for (ElementType element : set) {
    // Access element using 'element' variable
}

// Clearing the Set
set.clear();

// Creating an Immutable Set
Set<ElementType> immutableSet = Set.of(element1,
element2);
```

```java
// Union of Sets
Set<ElementType> unionSet = new HashSet<>(set1);
unionSet.addAll(set2);

// Intersection of Sets
Set<ElementType> intersectionSet = new
HashSet<>(set1);
intersectionSet.retainAll(set2);

// Difference of Sets
Set<ElementType> differenceSet = new
HashSet<>(set1);
differenceSet.removeAll(set2);
```



cheat sheet for commonly used methods and operations in the `Map` interface and its implementations in Java

~~~
Creating a Map:
```java
Map<KeyType, ValueType> map = new HashMap<>();
// HashMap
Map<KeyType, ValueType> map = new LinkedHashMap<>();
// LinkedHashMap (maintains insertion order)
Map<KeyType, ValueType> map = new TreeMap<>();
 // TreeMap (sorted)
```

Adding and Updating Entries:
```java
map.put(key, value); // Adds or updates an entry
with the specified key-value pair
map.putAll(otherMap); // Adds all entries from
another map to the current map
```

Retrieving Values:
ValueType value = map.get(key); // Retrieves the
value associated with the specified key
ValueType value = map.getOrDefault(key,
defaultValue); // Retrieves the value associated
with the specified key, or a default value if the
key is not present
Set<KeyType> keys = map.keySet(); // Retrieves a set
of all keys in the map
Collection<ValueType> values = map.values(); //
Retrieves a collection of all values in the map
~~~



~~~
Checking Existence:
boolean containsKey = map.containsKey(key);
// Checks if the map contains a specific key
boolean containsValue = map.containsValue(value);
// Checks if the map contains a specific value
boolean isEmpty = map.isEmpty();
// Checks if the map is empty

Removing Entries:
```java
ValueType removedValue = map.remove(key); // Removes
the entry with the specified key and returns its
value
map.clear(); // Removes all entries from the map

Size and Iteration:
```java
int size = map.size(); // Retrieves the number of
entries in the map
Set<Map.Entry<KeyType, ValueType>> entries =
map.entrySet(); // Retrieves a set of all entries in
the map
for (Map.Entry<KeyType, ValueType> entry :
map.entrySet()) {
    KeyType key = entry.getKey();
    ValueType value = entry.getValue();
    // Do something with key and value
}
```

Iterating with Java 8 Streams:
```java
map.forEach((key, value) -> {
    // Do something with key and value
});
~~~

cheat sheet for sorting and reversing elements in a collection using the `Collections` class in Java:

Sorting a Collection:
```java
List<T> list = new ArrayList<>(); // Replace T with the type of elements in the list
Collections.sort(list); // Sorts the list in ascending order
```

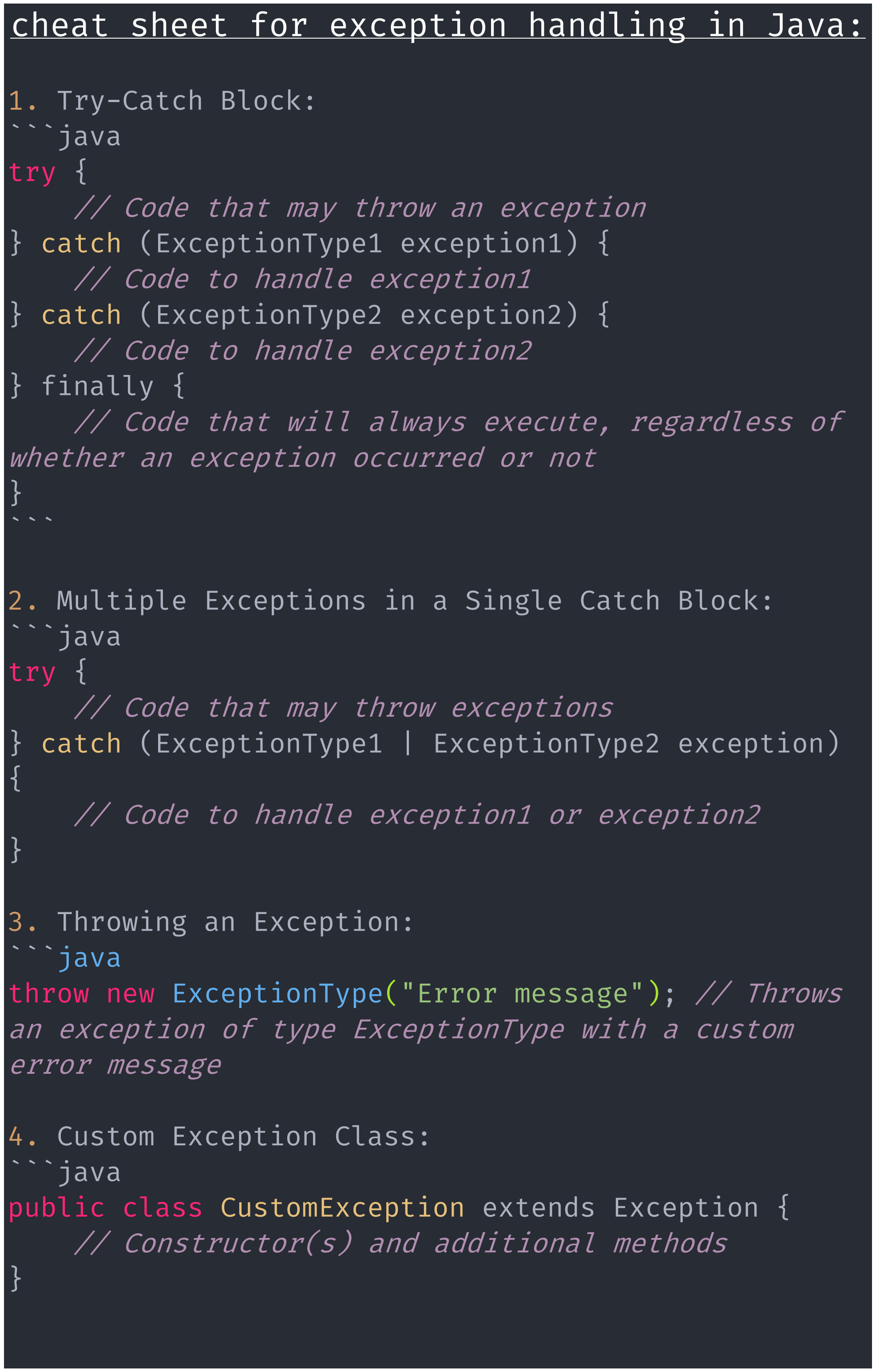
Sorting a Collection with a Custom Comparator:
```java
List<T> list = new ArrayList<>(); // Replace T with the type of elements in the list
Comparator<T> comparator = new CustomComparator();
// Replace CustomComparator with your own comparator implementation
Collections.sort(list, comparator); // Sorts the list using the custom comparator
```

Reversing the Order of a List:
```java
List<T> list = new ArrayList<>(); // Replace T with the type of elements in the list
Collections.reverse(list); // Reverses the order of elements in the list
```

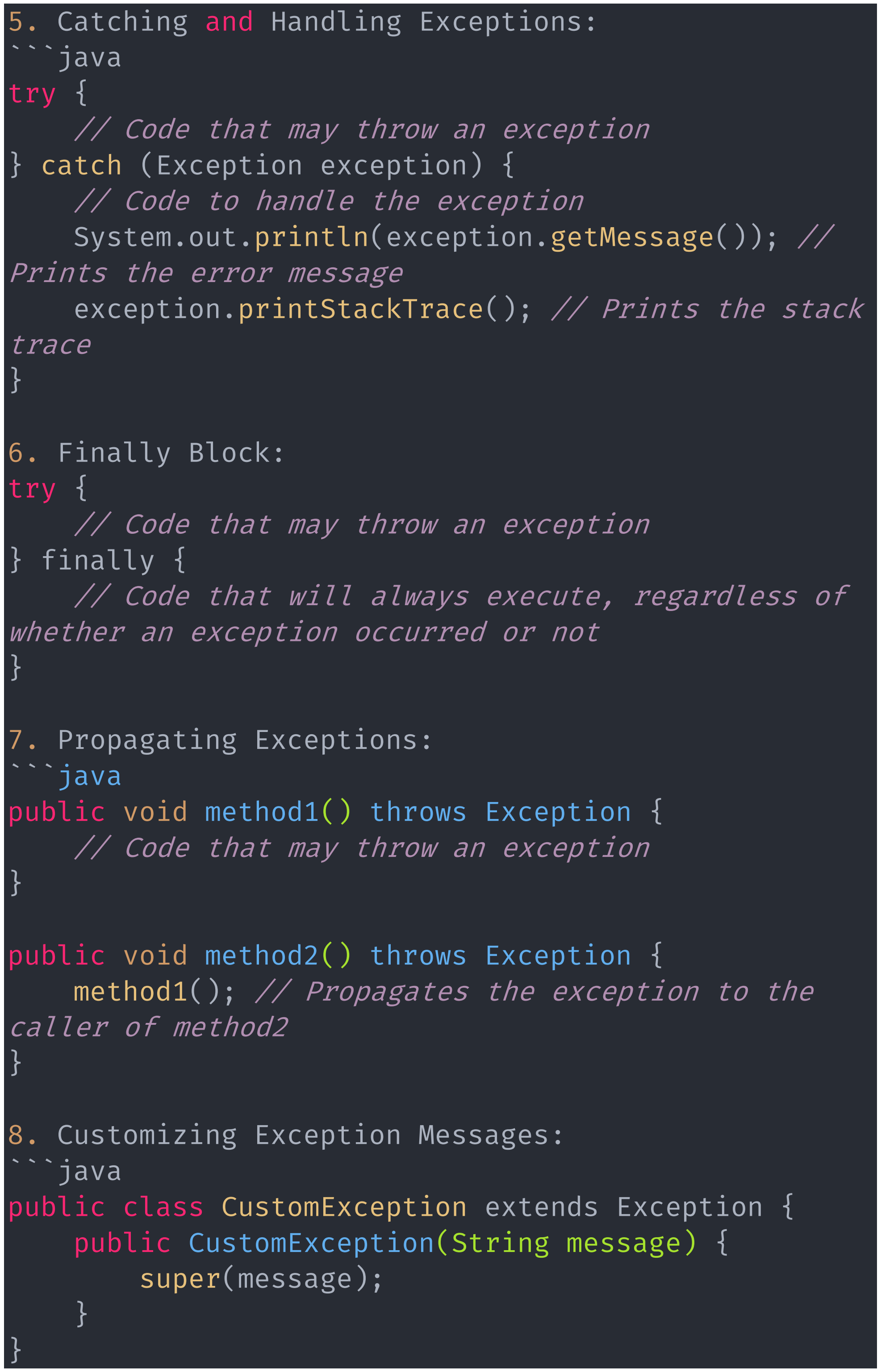
Shuffling the Elements in a List:
```java
List<T> list = new ArrayList<>(); // Replace T with the type of elements in the list
Collections.shuffle(list); // Randomly shuffles the elements in the list
```

It's important to note that the `Collections` class provides methods for sorting and manipulating elements in a `List` or `Collection`. If you're using other collection types like `Set` or `Queue`, you may need to convert them to a `List` before applying these methods.

Also, make sure that the elements in the collection implement the `Comparable` interface if you're using the `Collections.sort` method without a custom comparator. Otherwise, you'll encounter a `ClassCastException`.



~~~
cheat sheet for exception handling in Java:

1. Try-Catch Block:
```java
try {
    // Code that may throw an exception
} catch (ExceptionType1 exception1) {
    // Code to handle exception1
} catch (ExceptionType2 exception2) {
    // Code to handle exception2
} finally {
    // Code that will always execute, regardless of whether an exception occurred or not
}
```

2. Multiple Exceptions in a Single Catch Block:
```java
try {
    // Code that may throw exceptions
} catch (ExceptionType1 | ExceptionType2 exception)
{
    // Code to handle exception1 or exception2
}

3. Throwing an Exception:
```java
throw new ExceptionType("Error message"); // Throws an exception of type ExceptionType with a custom error message

4. Custom Exception Class:
```java
public class CustomException extends Exception {
    // Constructor(s) and additional methods
}
~~~



~~~
5. Catching and Handling Exceptions:
```java
try {
    // Code that may throw an exception
} catch (Exception exception) {
    // Code to handle the exception
    System.out.println(exception.getMessage()); // Prints the error message
    exception.printStackTrace(); // Prints the stack trace
}

6. Finally Block:
try {
    // Code that may throw an exception
} finally {
    // Code that will always execute, regardless of whether an exception occurred or not
}

7. Propagating Exceptions:
```java
public void method1() throws Exception {
    // Code that may throw an exception
}

public void method2() throws Exception {
    method1(); // Propagates the exception to the caller of method2
}

8. Customizing Exception Messages:
```java
public class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
~~~

cheat sheet for regular expressions (regex) in Java:

1. Basic Syntax:
   - `/pattern/`: Enclose the pattern within forward slashes.
   - `Pattern pattern = Pattern.compile("pattern");`: Compile the regex pattern into a `Pattern` object.

2. Metacharacters:
   - `.`: Matches any single character.
   - `^`: Matches the start of a line.
   - `$`: Matches the end of a line.
   - `[]`: Matches any character within the brackets.
   - `[^]`: Matches any character not within the brackets.
   - `|`: Matches either the pattern before or after the vertical bar.
   - `()`: Groups multiple patterns together.

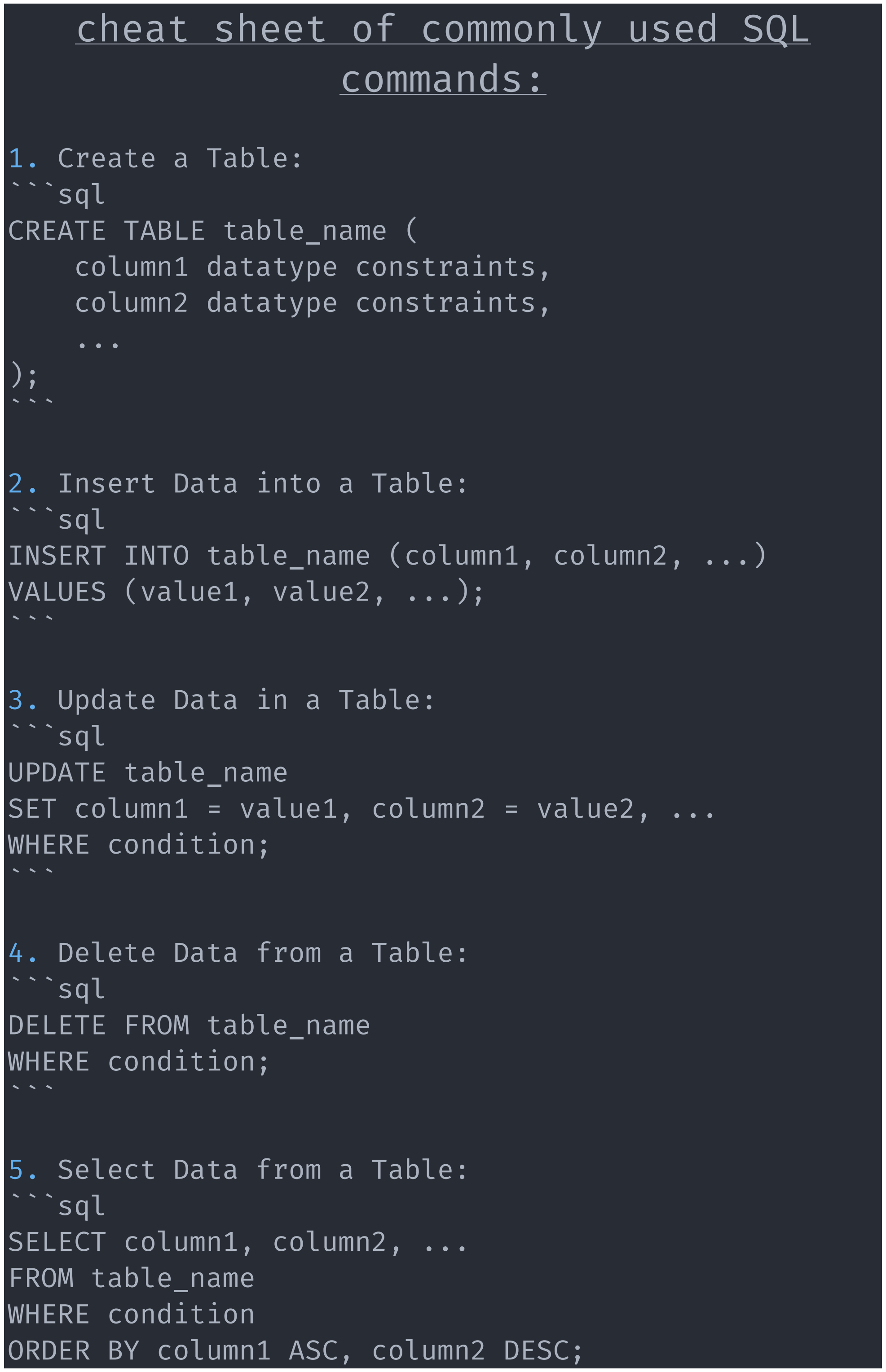
3. Quantifiers:
   - `*`: Matches zero or more occurrences of the preceding pattern.
   - `+`: Matches one or more occurrences of the preceding pattern.
   - `?`: Matches zero or one occurrence of the preceding pattern.
   - `{n}`: Matches exactly n occurrences of the preceding pattern.
   - `{n,}`: Matches at least n occurrences of the preceding pattern.
   - `{n,m}`: Matches at least n and at most m occurrences of the preceding pattern.

4. Predefined Character Classes:
   - `\d`: Matches a digit (0-9).
   - `\D`: Matches a non-digit.
   - `\w`: Matches a word character (alphanumeric and underscore).
   - `\W`: Matches a non-word character.
   - `\s`: Matches a whitespace character.
   - `\S`: Matches a non-whitespace character.

5. Boundary Matchers:
   - `\b`: Matches a word boundary.
   - `\B`: Matches a non-word boundary.
   - `^`: Matches the start of a line.
   - `$`: Matches the end of a line.

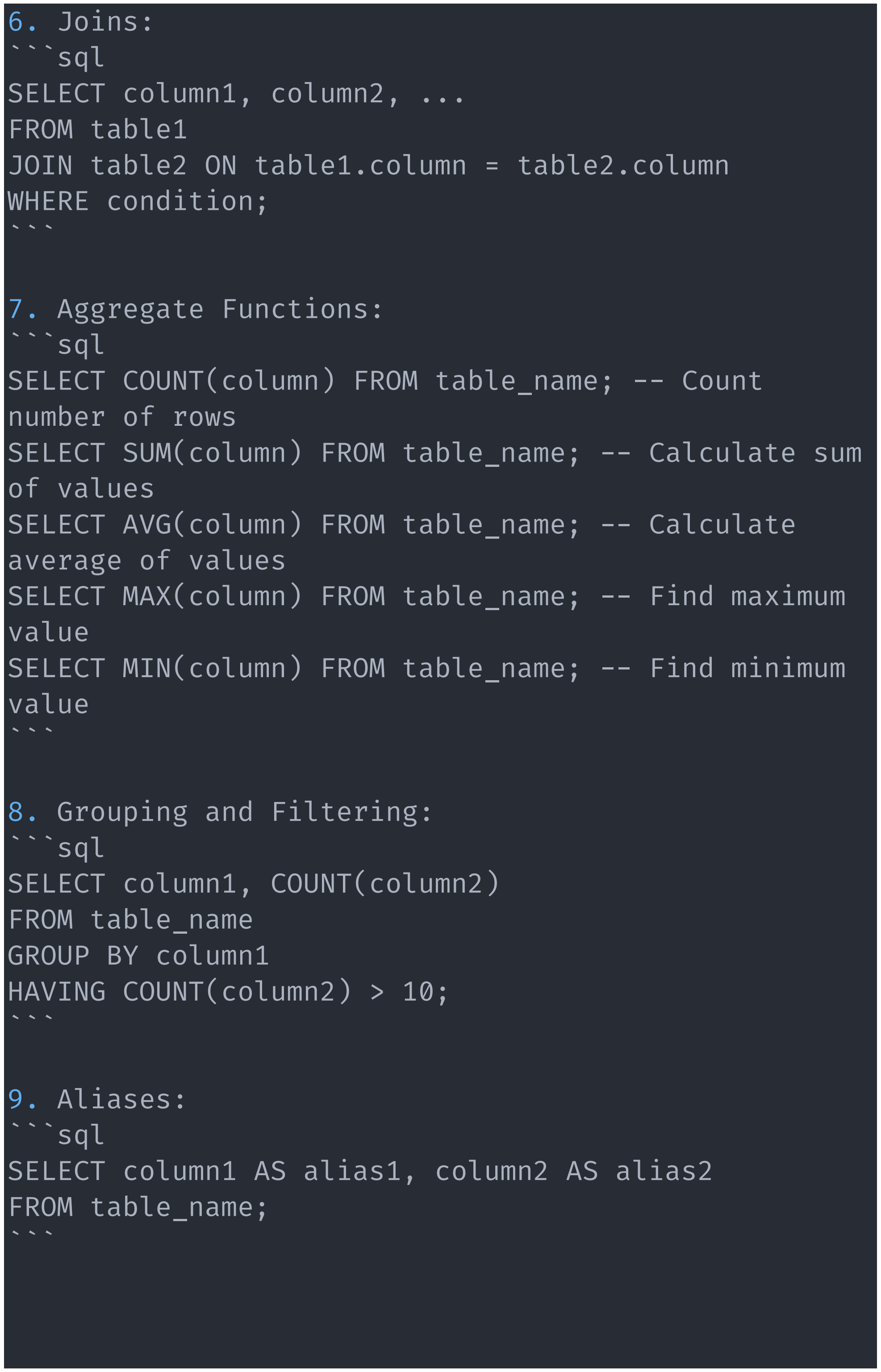
6. Flags:
   - `Pattern.CASE_INSENSITIVE`: Ignores case when matching.
   - `Pattern.MULTILINE`: Enables multiline matching.

7. Java Methods:
   - `Pattern.compile(regex)`: Compiles the regex pattern into a `Pattern` object.
   - `Matcher matcher = pattern.matcher(input)`: Creates a `Matcher` object for matching against the input string.
   - `matcher.matches()`: Checks if the entire input matches the pattern.
   - `matcher.find()`: Finds the next occurrence of the pattern in the input.
   - `matcher.group()`: Returns the matched substring.
   - `matcher.replaceAll(replacement)`: Replaces all occurrences of the pattern with the specified replacement.



cheat sheet of commonly used SQL commands:

~~~
1. Create a Table:
```sql
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
);
```

2. Insert Data into a Table:
```sql
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

3. Update Data in a Table:
```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

4. Delete Data from a Table:
```sql
DELETE FROM table_name
WHERE condition;
```

5. Select Data from a Table:
```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition
ORDER BY column1 ASC, column2 DESC;
~~~



~~~
6. Joins:
```sql
SELECT column1, column2, ...
FROM table1
JOIN table2 ON table1.column = table2.column
WHERE condition;
```

7. Aggregate Functions:
```sql
SELECT COUNT(column) FROM table_name; -- Count
number of rows
SELECT SUM(column) FROM table_name; -- Calculate sum
of values
SELECT AVG(column) FROM table_name; -- Calculate
average of values
SELECT MAX(column) FROM table_name; -- Find maximum
value
SELECT MIN(column) FROM table_name; -- Find minimum
value
```

8. Grouping and Filtering:
```sql
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1
HAVING COUNT(column2) > 10;
```

9. Aliases:
```sql
SELECT column1 AS alias1, column2 AS alias2
FROM table_name;
```
~~~

~~~
10. Sorting:
```sql
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 ASC, column2 DESC;
```

11. Limiting Results:
```sql
SELECT column1, column2, ...
FROM table_name
LIMIT 10;
```

12. Conditional Operators:
```sql
SELECT column1, column2, ...
FROM table_name
WHERE column1 = value1 AND column2 > value2 OR
column3 LIKE 'value%';
```

13. Creating Indexes:
```sql
CREATE INDEX index_name ON table_name (column1,
column2, ...);
```

14. Altering a Table:
```sql
ALTER TABLE table_name
ADD column datatype constraint;
```

15. Dropping a Table:
```sql
DROP TABLE table_name;
```
~~~

Prepared Statements Cheat Sheet:

```
Creating a Prepared Statement:
PreparedStatement statement =
connection.prepareStatement("SQL query");

Setting Parameters in a Prepared Statement:
java
statement.setDataType(parameterIndex, value);

Executing a Prepared Statement:
ResultSet resultSet = statement.executeQuery(); //
For SELECT queries
int rowsAffected = statement.executeUpdate(); // For
INSERT, UPDATE, DELETE queries

Retrieving Results from a Prepared Statement:
while (resultSet.next()) {
    // Process each row of the result set
    String value =
resultSet.getString("column_name"); // Retrieve
column value by column name
    // Or
    String value = resultSet.getString(columnIndex);
// Retrieve column value by column index
}

Closing Resources:
resultSet.close();
statement.close();
connection.close();
These are some basic SQL and Prepared Statement
operations. It's important to note that you should
always use parameterized queries (Prepared
Statements) to prevent SQL injection attacks.
```