Java Basics Cheat Sheet

1. What is Java?

Java is an object-oriented programming language used for building cross-platform applications. Its syntax is similar to C++ but simplified for ease of use.

2. Setting Up Java Environment

- **JDK (Java Development Kit)**: Needed to develop Java applications.
- **IDE (Integrated Development Environment)**: Popular ones are IntelliJ IDEA, Eclipse, and NetBeans.

3. Basic Structure of a Java Program

```
public class Main {

public static void main(String[] args) {

// This is the entry point of every Java program

  System.out.println("Hello, World!"); // Prints a message to the console

}

}
```

Explanation:

- **public class Main**: Defines a class named Main (the class name should match the file name).
- **public static void main(String[] args)**: The main method where the program starts. It must be present in every Java application.
- **System.out.println()**: Prints output to the console.

4. Variables and Data Types

Java is statically typed, meaning

```
int myNumber = 10; // Integer (whole number)
```

```
double myDecimal = 3.14; // Double (floating-point number)

char myLetter = 'A'; // Character

String myText = "Hello"; // String (sequence of characters)

boolean myBool = true; // Boolean (true or false)
```

Explanation:

- **int**: Whole numbers.
- **double**: Numbers with decimals.
- **char**: A single character, enclosed in single quotes.
- **String**: A sequence of characters, enclosed in double quotes.
- **boolean**: Can only be true or false.

5. Control Structures

Conditional Statements

- **if-else statement**: Executes different code blocks based on conditions.

```
int num = 5;

if (num > 0) {

 System.out.println("Positive number");

} else {

 System.out.println("Negative number or zero");

}
```

- **switch statement**: Executes one code block from many options based on a variable's value.

```
int day = 3;

switch (day) {

case 1:
```

```
 System.out.println("Monday");
 break;
case 2:
 System.out.println("Tuesday");
 break;
default:
 System.out.println("Another day");
}
```

Loops

- **for loop**: Repeats a block of code a set number of times.

```
for (int i = 0; i < 5; i++) {
 System.out.println(i);
}
```

- **while loop**: Repeats a block of code while a condition is true.

```
int i = 0;
while (i < 5) {
 System.out.println(i);
 i++;
}
```

6. Methods (Functions)

A method is a reusable block of code that performs a specific task.

```
public class Main {
// A method that adds two numbers
public static int addNumbers(int a, int b) {
 return a + b;
```

```
}



public static void main(String[] args) {

 int sum = addNumbers(3, 4);

 System.out.println("Sum: " + sum);

 }

}
```

Explanation:

- **Method declaration**: public static int addNumbers(int a, int b)
  - **public**: Access modifier, meaning the method can be called from anywhere.
  - **static**: Means the method belongs to the class and not to any object.
  - **int**: Return type, meaning this method returns an integer.
  - **addNumbers(int a, int b)**: Method name and parameters.
- **return a + b**: Returns the sum of the two numbers.

7. Classes and Objects

Java is object-oriented, meaning everything is grouped into objects. A **class** is a blueprint for creating objects (instances).

```
public class Car {

String model; // Attribute (property)



// Constructor (method to initialize objects)

public Car(String model) {

 this.model = model;

}
```

```
// Method (behavior)
public void startEngine() {
 System.out.println(model + " engine started.");
 }
}


public class Main {
 public static void main(String[] args) {
  Car myCar = new Car("Toyota"); // Create a new object
  myCar.startEngine(); // Call the method
 }
}
```

Explanation:

- **Class**: A template to define objects. The Car class has a model property and a method startEngine().
- **Object**: Instance of a class, e.g., myCar is an object of the Car class.
- **Constructor**: Special method called when creating an object. It initializes object properties.

8. Access Modifiers

Java provides access control to variables and methods:

- **public**: Accessible from anywhere.
- **private**: Only accessible within the class.
- **protected**: Accessible within the class and subclasses.

```
public class Person {
private String name; // Private variable
```

```
// Public method to access private variable
public void setName(String newName) {
 this.name = newName;
}

public String getName() {
return name;
 }
}
```

Explanation:

- **private**: The variable name is private, so it can only be accessed by public methods getName() and setName().

9. Inheritance

Inheritance allows one class to inherit fields and methods from another class.

```
class Animal {
 public void eat() {
  System.out.println("This animal eats food.");
  }
}

class Dog extends Animal { // Dog inherits from Animal
 public void bark() {
  System.out.println("The dog barks.");
 }
}
```

```
public class Main {
 public static void main(String[] args) {
  Dog myDog = new Dog();
  myDog.eat(); // Inherited method
  myDog.bark(); // Dog-specific method
  }
}
```

Explanation:

- **Inheritance**: The Dog class inherits the eat() method from the Animal class.

10. Exception Handling

Java uses try-catch blocks to handle exceptions (runtime errors).

```
try {
 int result = 10 / 0; // Will cause an ArithmeticException
} catch (ArithmeticException e) {
 System.out.println("Cannot divide by zero.");
}
```

Explanation:

- **try**: Wraps the code that might throw an exception.
- **catch**: Catches the exception and executes alternative code.

11. Importing Libraries

Java has extensive libraries. You can import them to use extra functionality.

```
import java.util.Scanner; // Import Scanner class



public class Main {
 public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  System.out.print("Enter your name: ");
  String name = scanner.nextLine();
  System.out.println("Hello, " + name);
 }
}
```

Object-Oriented Programming (OOP) in Java

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of “objects,” which are instances of classes. Java is a **pure object-oriented** language where everything revolves around the creation and interaction of objects. OOP provides four main principles: **Encapsulation, Inheritance, Polymorphism,** and **Abstraction**.

1. Encapsulation

Encapsulation is the practice of wrapping data (variables) and code (methods) together in a single unit called a class, and restricting access to certain components of the object to protect the data.

Example:

```
public class Person {

private String name; // private = restricted access

private int age;



// Getter method for name

public String getName() {

 return name;

}



// Setter method for name

public void setName(String newName) {

 this.name = newName;

}
```

```
// Getter method for age
public int getAge() {
 return age;
}

// Setter method for age
public void setAge(int newAge) {
 this.age = newAge;
}
}
```

Key Concepts:

- **Private** variables: The data is hidden from outside the class and can only be accessed through public getter and setter methods.
- **Encapsulation** helps achieve data hiding, and ensures that the internal representation of an object is protected from external modification.

2. Inheritance

Inheritance allows a new class (child/subclass) to acquire the properties and methods of an existing class (parent/superclass). This promotes code reusability.

Example:

```
class Animal {
public void eat() {
 System.out.println("This animal eats food.");
 }
}


class Dog extends Animal {
public void bark() {
 System.out.println("The dog barks.");
 }
}


public class Main {
public static void main(String[] args) {
 Dog dog = new Dog();
 dog.eat(); // Inherited from Animal class
 dog.bark(); // Dog-specific method
}
}
```

Key Concepts:

- **Super class** (parent): The class being inherited from (e.g., Animal).
- **Subclass** (child): The class that inherits the properties (e.g., Dog).
- Inheritance is used when classes share common properties or behavior, allowing for reuse and extending functionality.

3. Polymorphism

Polymorphism means "many forms." It allows methods to perform differently based on the object that is calling them. There are two types of polymorphism in Java:

1. **Compile-Time (Static) Polymorphism**: Achieved by **method overloading** (multiple methods with the same name but different parameters).
2. **Run-Time (Dynamic) Polymorphism**: Achieved by **method overriding** (a subclass provides a specific implementation of a method that is already defined in its superclass).

Compile-Time Polymorphism (Method Overloading):

```
class Calculator {

// Method to add two integers

 public int add(int a, int b) {

  return a + b;

}



// Overloaded method to add three integers

public int add(int a, int b, int c) {

 return a + b + c;

 }

}



public class Main {

 public static void main(String[] args) {

 Calculator calc = new Calculator();

  System.out.println(calc.add(2, 3)); // Calls method with 2 parameters

  System.out.println(calc.add(2, 3, 4)); // Calls method with 3 parameters

 }

}
```

Run-Time Polymorphism (Method Overriding):

```
class Dog extends Animal {
@Override
public void sound() {
  System.out.println("The dog barks");
 }
}


public class Main {
public static void main(String[] args) {
  Animal animal = new Dog(); // Polymorphic behavior
  animal.sound(); // Calls Dog's overridden method
 }
}
```

Key Concepts:

- **Method Overloading**: Same method name, different parameters (compile-time).
- **Method Overriding**: A subclass provides its own implementation of a method (run-time).
- Polymorphism helps improve flexibility and maintainability by allowing methods to be used in different forms.

4. Abstraction

Abstraction is the concept of hiding complex implementation details and showing only the essential features of an object. This is typically done using **abstract classes** or **interfaces** in Java.

Example of an Abstract Class:

```
class Animal {
public void sound() {
  System.out.println("This animal makes a sound");
  }
}


abstract class Animal {
// Abstract method (does not have a body)
public abstract void sound();


// Regular method
public void sleep() {
  System.out.println("This animal sleeps.");
  }
}


class Dog extends Animal {
public void sound() {
  System.out.println("The dog barks");
}
}


public class Main {
```

```
public static void main(String[] args) {
 Animal dog = new Dog(); // Abstract class reference, Dog object
 dog.sound(); // Calls the Dog's implementation of the sound method
 dog.sleep(); // Inherited from the Animal class
}
}
```

Example of an Interface:

```
interface Animal {
 public void sound(); // Interface method (does not have a body)
}


class Dog implements Animal {
public void sound() {
 System.out.println("The dog barks");
 }
}


public class Main {
public static void main(String[] args) {
 Dog dog = new Dog();
 dog.sound(); // Calls Dog's implementation of the sound method
 }
}
```

Key Concepts:

- **Abstract class**: Can have abstract (unimplemented) methods and concrete (implemented) methods. Cannot be instantiated directly.

- **Interface**: A contract that classes must follow. All methods in an interface are abstract (unimplemented) by default.
- **Abstraction** helps in separating the “what” from the “how” by providing simpler interfaces for more complex underlying functionality.

Why Use OOP?

- **Modularity**: Code can be divided into smaller, reusable components (classes and objects).
- **Code Reusability**: With inheritance, existing classes can be extended to add new functionality.
- **Flexibility**: Polymorphism allows methods to behave differently based on the object that calls them.
- **Maintainability**: Encapsulation and abstraction hide implementation details, making the system easier to maintain.

Conclusion

OOP provides a structured approach to writing code, which is easier to manage, reuse, and extend. The four pillars—**Encapsulation, Inheritance, Polymorphism,** and **Abstraction**—are fundamental to understanding how Java and other object-oriented languages work. By mastering these concepts, you can write more modular, flexible, and maintainable programs.