Core Java Cheat Sheet prepared by Haradhan Pal (Haradhan Automation Library)	
YouTube Channel Link	https://www.youtube.com/c/HaradhanAutomationLibrary?sub_confirmation=1
Java Features	
Simple	Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language.
Object-Oriented	Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules. Basic concepts of OOPs are: Object Class Inheritance Polymorphism Abstraction Encapsulation
Portable	Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
Secured	Java is best known for its security. With Java, we can develop virus-free systems.
Platform independent	Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.
Architecture-neutral	Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
High Performance	Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.
Multithreaded	A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.
Robust	Robust simply means strong. Java is robust because: There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore. There are exception handling and the type checking mechanism in Java. All these points make Java robust.
Interpreted	Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
JVM, JDK and JRE	
JVM	JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode. At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.

JRE	JRE (Java Runtime Environment) is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
JDK	The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.
Syntax Rules for Java	
Java Syntax Rules	Java is case sensitive language.
	First letter of Class Name should be in Upper case.
	Method names should start with lower case letter.
	Interface name should start with uppercase letter
	Package name should be in lowercase letter.
	Constant's name should be in uppercase letter.
	Java program file name should exactly match with class name.
	Java Program execution starts from main method, which is mandatory in every Java program.
	Every Statement should end with semi colon symbol.
	Code blocks enclosed with {}
Main and Syso Method	

Java Main Method	Main Function/Method: public static void main (String [] args) {
	}
	public – Access Modifier
	static – Non-Access Modifier
	void – Return Type (Returns nothing)
	Main - method name

Java syso Method	Sample Program (Syso Method): System.out.println(“My First Java Code”); System – Class (Pre-defined) out – This is an instance of PrintStream type, which is a public and static member field of the System class println – method “My First Java Code” – Message
Java Comments, Data Types, Variables and Operators	
Comments Syntax in Java:	Use // for Single line comments Use /* …… …….. …………. */ for Multiple lines comments
Primitive Data Types	i) Integer Types: 1) byte (8 bits); Example: byte a =10; 2) short (16 bits); Example: short a =1000; 3) Integer (32 bits); Example: int i = 10000; 4) long (64 bits); Example: long l =100000000000L; ii) Relational types (Numbers with decimal places) 5) float (32 bits); Example: float f = 1.23f or (float) 1.23; 6) double (64 bits); Example: double d =123.4567890; iii) Characters 7) Character (16 bits); Example: char c =’Z’ iv) Conditional 8) Boolean (1 bit); Example: boolean b = true;
Non-Primitive Data Types	Non-primitive or Reference data types in Java are Objects, Class, Interface, String and Arrays. Example: String str = “Java World”

Java Variable	a) Local variable: Local variable is declared in methods or blocks.
	b) Instance variable: Instance variables are declared inside the class but outside the body of the method. It is called instance variable because its value is instance specific and is not shared among instances.
	c) Class/Static variable: A Variable that is declared as static, It cannot be local. User can create a single copy of static variable and share among all the instances of the class.
	
	Example:
	class Test{
	int i1=10; //instance variable
	static int i2=20; //static/class variable
	void testmethod(){
	int i3=30; //local variable
	}
	}

Arithmetic Operators	1) Addition + (for Addition, String concatenation) 2) Subtraction – (for Subtraction, Negation) 3) Multiplication * 4) Division / 5) Modules % 6) Increment ++ 7) Decrement —
Relational Operators	1) == 2) != 3) > 4) >= 5) < 6) <=
Assignment Operators	1) Assignment Operator: = 2) Add and Assign: += 3) Subtract and assign: -= 4) Multiple and assign: *=
Logical Operators	1) Logical Not Operator ! 2) Logical And Operator && 3) Logical Or Operators ||
Java String and Array	
Java String	String is a sequence of characters written in double quotes. String may have Alphabets, Numbers and Special characters. The java.lang.String class provides many useful methods to perform operations on sequence of char values. An array of characters works same as Java string. String can be created using new keyword. **String Example:** String s1 = “Welcome to Java” char[] ch={'j','a','v','a'}; String s2=new String(ch); String s3=new String(“Java String");

Java Array	Array is a collection of similar type of elements that have a contiguous memory location. An array is a very common type of data structure wherein all elements must be of the same data type. Once defined, the size of an array is fixed and cannot increase to accommodate more elements, index starts from zero to n-1. We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically. There are two types of array. Single Dimensional Array – Example: int [] array1 = {1, 2, 3, 4}; Multidimensional Array – Example: int [] [] array2 = {{1, 3, 5}, {2, 4, 6}};
Java Loops	
For Loop	The Java for loop is a control flow statement that iterates a part of the program multiple times. For loop repeats a block of statements for a specified number of times. If the number of iteration is fixed, it is recommended to use for loop. Syntax: for (startValue; endValue; increment/decrement){ Statement(s) }

Enhanced For Loop	In Java, there is another form of for loop (in addition to standard for loop) to work with arrays and collection, the enhanced for loop.
	Enhanced for loop is also known as for-each loop which reduces the code significantly and there is no use of the index or rather the counter in the loop.
	Syntax of enhanced for loop:
	for(declaration : expression)
	{
	// Statements
	}
	
	char [] obj = {'j','a', 'v', 'a'};
	for (char c: obj)
	{
	System.out.println(c);
	}

While Loop	A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true. Syntax: Initialization; while (Condition){ Statement(s); increment/decrement; }
Do while Loop	The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. It executes a block of statements at least once irrespective of the condition. First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control gets transferred to the “do” else it jumps to the next statement after do-while. Syntax: Initialization; do { Statement(s); increment/decrement; } while (Condition);
Conditional Statement in Java	

If	if statement is used only to specify a block of Java code to be executed if a condition is met (true).
	 if(condition){
	  Statement(s);
	}

If else	An if statement can be followed by an optional else statement, else statement is used to specify a block of code to be executed if the condition is not met (false).
	 if(condition) {
	   Statement(s);
	}
	else {
	   Statement(s);
	}

if else if ladder	else if statement is used to specify a new condition when first condition is false.
	
	if(condition_1) {
	   //execute this statement in case condition_1 is true
	   Statement(s);
	}
	else if(condition_2) {
	   //execute this statement in case condition_1 is not met but condition_2 is true
	   Statement(s);
	}
	else if(condition_3) {
	   //execute this statement in case condition_1 and condition_2 are not met but condition_3 is true
	   Statement(s);
	}
	.....
	else {
	  //execute this statement in case none of the above conditions are true
	   Statement(s);
	}

Switch	switch(expression) {
	   case value :
	      // Statements
	      break; // optional
	     case value :
	      // Statements
	      break; // optional
	.
	.
	     default : // Optional
	      // Statements
	}

Different Type of Exceptions in Java	
Arithmetic Exception	It is thrown when an exceptional condition has occurred in an arithmetic operation.
ArrayIndexOutOfBound Exception	It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
ClassNotFoundExceptio n	This Exception is raised when we try to access a class whose definition is not found

FileNotFoundException	This Exception is raised when a file is not accessible or does not open.
IOException	It is thrown when an input-output operation failed or interrupted
InterruptedException	It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
NoSuchFieldException	It is thrown when a class does not contain the field (or variable) specified
NoSuchMethodException	It is thrown when accessing a method which is not found.
NullPointerException	This exception is raised when referring to the members of a null object.
NumberFormatException	This exception is raised when a method could not convert a string into a numeric format.
RuntimeException	This represents any exception which occurs during runtime.
StringIndexOutOfBoundsException	It is thrown by String class methods to indicate that an index is either negative than the size of the string
Java OOPS Concept	

Class	A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
	Example: Class: Animal; Object: Tiger, Elephant, Lion, Human, Cow, Dog etc.
	`class <class_name> {`
	`    field;`
	`    method;`
	`}`

Object	An object in Java is the physical as well as logical entity whereas a class in Java is a logical entity only.
	An entity that has state and behavior is known as an object e.g. chair, table, ball, cycle,, bike car etc.
	`//Defining a Student class.`
	`class Student{`
	`//defining fields`
	`int rollno= 1;`
	`String name = “Raj";`
	`//creating main method inside the Student class`
	`public static void main(String args[]){`
	`//Creating an object or instance`
	`Student student1=new Student();  //creating an object of Student`
	`//Printing values of the object`
	`System.out.println(student1.rollno);  //accessing member through reference variable`
	`System.out.println(student1.name);`
	`   }`
	`}`

Inheritance	Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPS (Object Oriented Programming System).
	Using Inheritance we can create Classes that are built in upon existing classes.
	When we Inherit from an existing class, then we can reuse Methods and fields from Parent class and we can add new methods and fields.
	The class where the class members are getting Inherited is called as Super class/Parent class/Base class.
	The class to which the class members are getting Inherited is called as Sub class/Child class/Derived class.
	The Inheritance between Super class and Sub class is achieved using “extends” keyword.
	1) Single Inheritance
	Example: public Class ClassB extends Class A { } 2) Multi level Inheritance
	Example: public Class ClassB extends ClassA {
	public Class ClassC extends ClassB {
	} }

Polymorphism	Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism derived from two Greek words, Poly-means Many and Morphs - means ways; So polymorphism means many ways. There are two types of Polymorphism is available in Java. a) Compile Time Polymorphism (Method Overloading) b) Run-time Polymorphism (Method Overriding)

Method Overloading	Two are more methods having same name in the same class but they differ in following ways.
	a) Number of Arguments
	b) Type of Arguments
	Example for Method Overloading:
	public class Method Overloading {
	public void add(int a, int b){
	System.out.println(a+b);
	}
	public void add(int a, int b, int c){
	System.out.println(a+b+c);
	}
	public void add(double a, double b){
	System.out.println(a+b);
	}
	public static void main(String[] args) {
	Method Overloading obj = new Method Overloading();
	obj.add(100, 200);
	obj.add(15, 25, 35);
	obj.add(101.234, 23.456);
	}

Method Overriding	If subclass (child class) has the same method as declared in the parent class, it is known
	as method overriding in Java.
	When a method in a subclass has the same name, same parameters or signature and
	same return type(or sub-type) as a method in its super-class, then the method in the
	subclass is said to override the method in the super-class.
	Lets take a simple example to understand this. We have two classes: A child class Girl and
	a parent class Human. The Girl class extends Human class. Both the classes have a
	common method void eat(). Girl class is giving its own implementation to the eat()
	method or in other words it is overriding the eat() method.
	Example:
	class Human{
	//Overridden method
	public void eat()
	{
	System.out.println("Human is eating");
	}}
	class Girl extends Human{
	//Overriding method
	public void eat(){
	System.out.println("Girl is eating");
	}
	public static void main(String args[]) {
	Girl obj = new Girl();
	//This will call the child class version of eat()
	obj.eat();
	}
	} //Output: Girl is eating
Abstraction	Abstraction is a process of hiding implementation details and showing only functionality
	to the user.
	In another way it shows important things to the user and hides internal details, for
	example, sending SMS where you type the text and send the message. One don't know
	the internal processing about the message delivery.
	Abstraction focuses on what the Object does instead of how it does.
	A class which is declared with the abstract keyword is known as an abstract class in Java. It
	can have abstract and non-abstract methods (method with the body).
	From Class (Concrete class or Abstract Class) to Class we use “extends” keyword
	Example:
	abstract class Animal{
	abstract void run();
	}
	class Tiger extends Animal{
	void run()
	{System.out.println(“Tiger is running");
	}
	public static void main(String args[]){
	Tiger obj = new Animal();
	obj.run();
	} }

Interface	Interface is a Java type definition block which is 100% abstract All the Interface methods by default public and abstract. static and final modifiers are not allowed for interface methods. In Interfaces variables are public, static, and final by default. Interface can not be instantiated, and it does not contain any constructors. Interface can be used using “implements” keyword for a Class. A class that implements an interface must implement all the methods declared in the interface. From Interface to Interface, we use “extends” keyword Example: `public interface Bike{ public void engine(); public void wheels(); public void seat(); }`

Encapsulation	Encapsulation is one of the four fundamental OOPS concepts. The other three are inheritance, polymorphism, and abstraction.
	Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.
	In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
	Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.
	
	`public class TestEncapsulation{`
	`private String animalname;   //private data member`
	`public String getName(){`
	`return  animalname;  //getter method for  animalname`
	`}`
	`public void setName (String  animalname){`
	`this.animalname = animalname   //setter method for  animalname`
	`}`
	`}`

Important Java keywords	

Static	The static keyword in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.
	When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
	It is possible to create a class members (variables and methods) that can be accessed and invoked without creating an instance of the class. In order to create such a member, we have to use static keyword while declaring a class member.
	
	`public class TestStaticVariable {`
	`static int m = 10; // static variable`
	`int n = 5; // non static variable`
	`    public static void main(String[] args) {`
	`        TestStaticVariable tsv = new TestStaticVariable();`
	`        System.out.println(tsv.n);`
	`        System.out.println(m);`
	`    }`
	`}`

Final	final keyword is used in different contexts. First of all, final is a non-access modifier applicable only to a variable, a method or a class.
	The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable but we cannot change the value of a final variable once assigned. Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.
	A final method cannot be overridden by any subclasses. The final modifier prevents a method from being modified in a subclass. The main intention of making a method final would be that the content of the method should not be changed by any outsider.
	The main purpose of using a class being declared as final is to prevent the class from being subclassed. If a class is marked as final then no class can inherit any feature from the final class.
	class Test
	{ public static void main(String args[])
	   {
	       final int i; // local final variable
	       i = 20;
	//i=50; It will throw error
	       System.out.println(i);
	   } }

Super	The super keyword in Java is a reference variable which is used to refer immediate parent class object.
	Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.
	Super keyword can be used at variable, method and constructor level.
	Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.
	
	class Animal{
	String Name="Elephant";
	}
	class Dog extends Animal{
	String Name="Bullet";
	void printName(){
	System.out.println(Name);//prints Name of Dog class
	System.out.println(super.Name);//prints Name of Animal class
	}}
	class TestSuper{
	public static void main(String args[]){
	Dog d=new Dog();
	d.printName();
	}}

throw	The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. The throw keyword is mainly used to throw custom exceptions.
	Throw is used within the method
	Syntax:
	throw Instance
	Example: throw new ArithmeticException(“ Any number divided by zero");
	class Testthrow {
	public static void divideNumberByZero() {
	throw new ArithmeticException("Trying to divide any number by 0");
	}
	public static void main(String[] args) {
	divideNumberByZero();
	}
	}

throws	The throws keyword is used declare the list of exceptions which may be thrown by that method or constructor.
	The throws is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.
	There are many exception types available in
	Java: ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException etc.
	The throws keyword can be followed by one more exception class, separated by commas.
	Syntax:
	public static void testmethodname( ) throws FileNotFoundException, ConnectionException {
	   //code
	}
	void testMethod() throws Exception1, Exception2 {
	   if (an exception occurs) {
	      throw new Exception1();
	   }
	   if (another exception occurs) {
	      throw new Exception2();
	   }
	}