Programs		
Compiled	Translated to Machine Code	Javac (.class)
Interpreted	Codes directly Read and Executed	Java to (.java)

Programming Languages: Typed Property		
Dynamic	Variable can hold values of different unrelated types	Python, Javascript
Static	Variable types are declared, and only hold values of that type and subtypes	Java, C
Strong	Enforce strict rules in type system, ensuring type safety (catch during compile time)	Java
Weak	Allow typecasting that changes interpretation of byte	C
Primitive	Predetermined values of the language, never sharing value with each other (byte, short, int, long, float double, char)	
Reference		

Java Primitive Type Format Placeholders & Sizes (bits)								
boolean	%b	1	char	%c	16	byte	%d	8
short	%d	16	int	%d	32	long	%d	64
float	%0.1f	32	double	%0.1f	64			
$byte <: short <: int <: long <: float <: double$								
$char <: int$								

Subtype Properties	
Reflexive	For any type S, we have $S <: S$
Transitive	$(S <: T) \wedge (T <: U) \rightarrow S <: U$
Anti-Symmetric	$(S <: T) \wedge (T <: S) \rightarrow S = T$

Object-Oriented Programming Principles	
Abstraction	Hides internal details (Method, Variable Names) Composite Data Type (Struct, Class, Object)
Encapsulation	Bundles Data and Methods in Class (Private Data, Public Method)
Inheritance	IS-A relationship, extends the parent class, sharing a set of properties
Polymorphism	Refer to "Dynamic Binding" table, allow Override, Specifically, Inclusion Polymorphism

OOP Principles Implications	
Special Reference Value: Null	Variable uninitialized will take the value: null Refers to a non-existent instance
Keywords	Refer to "Keywords" table, determines characteristics of the Class, Method or Field
Abstraction Barrier	Implementer: Implements Codes Client: Use Codes (No idea about implementation)
Reduced Code Complexity	Functions group a set of actions and codes, hiding implementation, simplifying the code, reduce repetition
Data Hiding	Private: Accessible only withing Class Public: Accessible in and outside of Class

Tell, Don't Ask	Refer to "Tell, Don't Ask" table
Class, Fields, Methods, Interface	Refer to the "Class, Fields, Methods, Interface" table
Composition	A Class as a Class Field (ie Circle has Point) Objects can be shared (Hence Accessors for Objects can be dangerous), HAS-A Relationship
Run-Time Type	The exact type of the object the variable points to, must be a Subtype of Compile Type
Liskov Substitution Principle (LSP)	When overriding, maintain the spirit of the method, "Let ϕ(x) be a property provable about objects x of type T. Then ϕ(y) should be true for objects y of type S where S<:T."
Type Conversion	Narrowing: S to T where S <: T, lose information Widening: T to S where S <: T

Keywords	
public	Accessible to all classes
private	Inaccessible to all classes
this	Points to the instance / object itself, Only relevant to initialised instances ie cannot access static value
super	Points to the instance immediate parent
static	Makes variable / method a Class Method (ie shared in Class)
@Override	Explicitly declares overriding method of a parent, throws "OverrideError" if parent method not found
final	Fields: Can't Re-Assigned Method: Can't be Overriden Class: Can't be inherited
abstract (Concrete: !abstract)	Class: Can contain abstract method, cannot be instantiated Methods: No body declared, Must be declared in concrete subclasses
try catch (E e) finally	try: start block, stops upon Exception thrown to catch catch (E e): runs block if Exception thrown in try matches E finally: runs block after try or catch, executed even after return or throw is called
@Suppress Warnings("")	Tells compiler to ignore warnings, used with String arguments "rawtypes" or "unchecked" to ignore the respective warnings. Good practice to add comments as to why the warnings are suppressed ie, why it is safe
@SafeVarags	Tells compiler that the generic arguments are safe and ignore unchecked warning

Tell, Don't Ask	
Description	Tell Class to do the work, not ask for the data and manipulate them
Accessors	Getter Methods: returns the data
Mutators	Setter Methods: changes the data
DANGER	Able to edit the values (without verification) Applicable to Accessors when data is an Object

Class, Fields, Methods, Interface	
Class & Interface Declaration Order	Public/private? static? final? name (extends Class) (implement Interface)
Fields & Methods Declaration Order	public/private? static? final? return name
void Return Type	Special Type of Methods to return nothing
Interface	Collection of implicitly Abstract Methods
Static Class Field	A field shared within a class: Belongs to the Class (Different from Static Typed: Property of Programming Language)
Non-Static Class Field	Fields only initialised and accessible from the instance of the class holding it: Belongs to the Instance
Static Class Methods	A method shared within a class: Belongs to the Class, Unable to access Non-Static Class Fields
Non-Static Methods	A method only initialised and accessible from the stance of the class holding it: Belongs to the Instance, Able to access Static and Non-Static Class Fields

main method	
Descriptions	Entry point to the program
Declaration	public static final void main(String[] args) {}

Heap & Stack (By Java Virtual Machine, JVM)	
Method Area	Stores code for the methods
Metaspace	Stores meta information about classes
Heap	Stores Dynamically Allocated Objects
Stack	For local variables and call frames Last-In-First-Out (LIFO)
Empty ∅ (null)	Denote uninitialized variables
Pointers	Points to the Objects in Heap, Primitives are stored directly to the variables
Garbage Collector	Checks for unreferenced objects on heap and cleans up the memory automatically

Object Class	
Immutable Objects	Its state cannot change after construction
Implicit Inheritence	Classes that does not extend another class inherits from Object implicitly
toString() Method	Converts reference object to a String object, called implicitly by Java
equals(Object obj) Method	Check if Object input refers to the same Object instance
@Override	Override parent method with the same method descriptor
Method Signature	Method Name; number, type and order of Parameters
Method Descriptor	Method Signature + Return Type
Method Overloading	Methods with same name, differing Method Signature

Wrapper Classes					
Wrapper Class		Encapsulate a primitive type, Immutable			
Auto-boxing & Unboxing		Auto-Boxing: primitive to Wrapper Unboxing: Wrapper to primitive			
Tradeoff vs Primitive		Performance & Memory Allocation			
byte	Byte	short	Short	int	Integer
long	Long	float	Float	double	Double
char	Character	boolean	Boolean		

Dynamic Binding (Late Binding / Dynamic Dispatch)	
Description	Method of same signature invoked is decided based on run-time type of instance calling the method
Method Invocation	Method Descriptor: Compile Time (Target Type, Param Number, Type, Order) Method Implementation: Run-Time (Target Type)
Example	Given Class A, with methods equals(A), override equals(Object): Object.equals(Object / A)⇒Object::equals(Object) (Object obj = A).equals(Object / A)⇒A::equals(Object) A.equals(Object obj = Object / A)⇒A::equals(Object) A.equals(A)⇒A::equals(A)
Class Method Invocation	Does not support Dynamic Binding, resolved statically during compile time

Type Casting	
Description	Ask compiler to trust that instance has a run-time type of a subtype
Relationship	At Compile Time: must have subtype relationship ie (S) T, then S<:T or T<:S At Run Time: Referenced Instance must be subtype of casting type
Run-Time Class Mismatch	When the Run Time relationship stated above is not met
Casting to Interface	Class may implement interface despite undeclared, Always Compiles

Variances	
Covariant	$S <: T \Rightarrow C(S) <: C(T)$
Contravariant	$S <: T \Rightarrow C(T) <: C(S)$
Invariant	Neither Covariant nor Contravariant
Example	Array: Covariant (Integer <: Object⇒Integer[] <: Object[]) Generics: Invariant (Seq<Integer> not a variant of Seq<Object>, vice versa) Upper Bounded Wildwards: Covariant (Seq<? extends Integer> <: Seq<? extends Object>) Lower Bounded Wildcards: Contravariant (Seq<? super Object> <: Seq<? super Integer>)

Exception	
Exception	Subclass of Throwable Errors managed with try / catch / finally

Unchecked Exception	Caused by programmer's error, subclass of RuntimeException Eg. IllegalArgumentException, NullPointerException, ClassCastException
Checked Exception	Out of programmer's error, user error, Must be handled or cannot compile Eg. FileNotFoundException, InputMismatchException
Method Throwing Exception	<method descriptor> throws Exception { throw new Exception(message?); }
Catch Exceptions to Clean Up	Handle the Exception appropriately based on the Exceptions caught
BAD: Pokemon Exception Handling	Catching all Exceptions ie catch (Exception e)
BAD: Overreacting	Exiting program when Exception thrown, prevents calling function from cleaning up resources, worse, exiting program silently ie without comment
BAD: Breaking Abstraction Barrier	Leaking information of implementation behind abstraction barrier
BAD: Use Exception as Control Flow Mechanism	Intentionally throwing an Exception in try block to go to a catch block, may end up catching a valid but unintended Exception
Error Class	For situations where program should terminate as generally no way to recover from error, typically no need to create or handle such errors Eg. OutOfMemoryError, StackOverflowError

Generics	
Generic Types	Takes on other types as type parameters Eg. <T>
Type Arguments	To put into type parameters during instantiation Eg. <String>, <S>, <>
Parameterized Type	Instantiated Generic Type
Generic Classes	Eg. Pair<S, T>, Array<T>
Generic Methods	Declare generic type before return type, parameter type is scoped within the whole method Eg. <T> T getFirstElem(T[] tArr) { return tArr[0]; }
Bounded Type Parameters	Sets upper boundary of generic type with extends Eg. <T extends Object>
Type Erasure	Due to code sharing approach of Java, Java erases type parameters and type arguments during compilation Transform Generic Classes or Methods to type parameters upper bound, then cast to type argument Eg. Pair<String, Integer>("", 1).getFirst() to (String) Pair("", 1).getFirst()
Generics & Arrays	Generics and Arrays can't mix, Arrays are reifiable, but Generics are non-reifiable due to type erasure Eg. new Pair<String, Integer>[int]
Rule of generic array	Generic array *declaration* is fine but generic array *instantiation* is not Eg. T[] arr

Heap Pollution	A term that refers to the situation where a variable of a parameterized type refers to an object that is not of that parameterized type
Reifiable Type	A type where full type information is available during run-time
Seq Class	Wrapper class for array to allow safer type erasure
Raw Type	A generic type used without type arguments Eg. Seq
instanceof	Checks type of instance Eg. String instanceof Object
Suppress Warnings	unchecked: Compiler unable to guarantee type erasure is safe rawtype: Use of rawtypes, can refer to any instance of any type
-Xlint:	Use with unchecked / rawtype to get warning message
Wildcards	Denoted as ?, can be used as a substitute for any type, Can be interpreted as a set of any type
Upper Bounded Wildcards	Denoted as <? extends Class>, the wildcard will only accept substitutes for subclasses of Class, Can be interpreted as a set of types that are subclasses of Class, is covariant
Lower Bounded Wildcards	Denoted as <? super Class>, the wildcard will only accept substitutes for superclasses of Class, Can be interpreted as a set of types that areeee subclasses of Class, is contravariant
Unbounded Wildcards	Denoted as <?>, is supertype of every parameterized type of its class, allow flexibility for methods to accept all types, An appropriate substitute for Rawtypes Eg. Class<AnyType> <: Class<?>

Type Inference	
Description	Decides what type the output will be
Considerations	Output Type must satisfy: Subtype of Target type Type Bound of Method Type Parameter Type Bound of Return Type Parameter Type Bound of Argument Type Parameter Given a type range, pick most specific type that can satisfy all in range
Examples	Type1 <: T <: Type2, pick T = Type1 Type1 <: T, pick T = Type1 T <: Type2, pick T = Type2