Java Introduction

**Define “computer program” and programming.**

* Computers execute very basic instructions in rapid succession.
* A computer program is a sequence of instructions and decisions.
* Programming is the act of designing and implementing computer programs.

**Describe the components of a computer.**

* The central processing unit (CPU) performs program control and data processing.
* Storage devices include memory and secondary storage.

**Describe the process of translating high-level languages to machine code.**

* Java was originally designed for programming consumer devices, but it was first successfully used to write Internet applets.
* Java was designed to be safe and portable, benefiting both Internet users and students.
* Java programs are distributed as instructions for a virtual machine, making them platform-independent.
* Java has a very large library. Focus on learning those parts of the library that you need for your programming projects.

**Become familiar with your Java programming environment.**

* Set aside some time to become familiar with the programming environment that you will use for your class work.
* An editor is a program for entering and modifying text, such as a Java program.
* Java is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.
* The Java compiler translates source code into class files that contain instructions for the Java virtual machine.
* Develop a strategy for keeping backup copies of your work before disaster strikes.

**Describe the building blocks of a simple program.**

* Classes are the fundamental building blocks of Java programs.
* Every Java application contains a class with a main method. When the application starts, the instructions in the main method are executed.
* Each class contains declarations of methods. Each method contains a sequence of instructions.
* A method is called by specifying the method and its arguments.
* A string is a sequence of characters enclosed in quotation marks.

**Classify program errors as compile-time and run-time errors.**

* A compile-time error is a violation of the programming language rules that is detected by the compiler.
* A run-time error causes a program to take an action that the programmer did not intend.

**Write pseudocode for simple algorithms.**

* Pseudocode is an informal description of a sequence of steps for solving a problem.
* An algorithm for solving a problem is a sequence of steps that is unambiguous, executable, and terminating.

Fundamental Data Types

**Declare variables with appropriate names and types.**

* A variable is a storage location with a name.
* When declaring a variable, you usually specify an initial value.
* When declaring a variable, you also specify the type of its values.
* Use the int type for numbers that cannot have a fractional part.
* Use the double type for floating-point numbers.
* By convention, variable names should start with a lowercase letter.
* An assignment statement stores a new value in a variable, replacing the previously stored value.
* The assignment operator = does *not* denote mathematical equality.
* You cannot change the value of a variable that is defined as final.
* Use comments to add explanations for humans who read your code. The compiler ignores comments.

**Write arithmetic expressions in Java.**

* Mixing integers and floating-point values in an arithmetic expression yields a floating-point value.
* The ++ operator adds 1 to a variable; the −− operator subtracts 1.
* If both arguments of / are integers, the remainder is discarded.
* The % operator computes the remainder of an integer division.
* The Java library declares many mathematical functions, such as Math.sqrt (square root) and Math.pow (raising to a power).

You use a cast (*typeName*) to convert a value to a different type.

**Write programs that read user input and print formatted output.**

* Java classes are grouped into packages. Use the import statement to use classes from packages.
* Use the Scanner class to read keyboard input in a console window.
* Use the printf method to specify how values should be formatted.
* The API (Application Programming Interface) documentation lists the classes and methods of the Java library.

**Carry out hand calculations when developing an algorithm.**

* Pick concrete values for a typical situation to use in a hand calculation.

**Write programs that process strings.**

* Strings are sequences of characters.
* The length method yields the number of characters in a string.
* Use the + operator to *concatenate* strings; that is, to put them together to yield a longer string.
* Whenever one of the arguments of the + operator is a string, the other argument is converted to a string.
* Use the next method of the Scanner class to read a string containing a single word.
* String positions are counted starting with 0.
* Use the substring method to extract a part of a string.

Methods

**Understand the concepts of methods, arguments, and return values.**

* A method is a named sequence of instructions.
* Arguments are supplied when a method is called.
* The return value is the result that the method computes.

**Be able to implement methods.**

* When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result.
* Method comments explain the purpose of the method, the meaning of the parameter variables and return value, as well as any special requirements.

**Describe the process of parameter passing.**

* Parameter variables hold the arguments supplied in the method call.

**Describe the process of returning a value from a method.**

* The return statement terminates a method call and yields the method result.
* Turn computations that can be reused into methods.

**Design and implement methods without return values.**

* Use a return type of void to indicate that a method does not return a value.

**Develop methods that can be reused for multiple problems.**

* Eliminate replicated code or pseudocode by defining a method.
* Design your methods to be reusable. Supply parameter variables for the values that can vary when the method is reused.

**Apply the design principle of stepwise refinement.**

* Use the process of stepwise refinement to decompose complex tasks into simpler ones.
* When you discover that you need a method, write a description of the parameter variables and return values.
* A method may require simpler methods to carry out its work.

**Determine the scope of variables in a program.**

* The scope of a variable is the part of the program in which it is visible.
* Two local or parameter variables can have the same name, provided that their scopes do not overlap.

**Understand recursive method calls and implement simple recursive methods.**

* A recursive computation solves a problem by using the solution of the same problem with simpler inputs.
* For a recursion to terminate, there must be special cases for the simplest inputs.
* The key to finding a recursive solution is reducing the input to a simpler input for the same problem.
* When designing a recursive solution, do not worry about multiple nested calls. Simply focus on reducing a problem to a slightly simpler one.

Input / Output and Exception Handling

**Develop programs that read and write files.**

* Use the Scanner class for reading text files.
* When writing text files, use the PrintWriter class and the print/println/printf methods.
* Close all files when you are done processing them.

**Be able to process text in files.**

* The next method reads a string that is delimited by white space.
* The Character class has methods for classifying characters.
* The nextLine method reads an entire line.
* If a string contains the digits of a number, you use the Integer.parseInt or Double.parseDouble method to obtain the number value.

**Process the command line arguments of a program.**

* Programs that start from the command line receive the command line arguments in the main method.

**Use exception handling to transfer control from an error location to an error handler.**

* To signal an exceptional condition, use the throw statement to throw an exceptionobject.
* When you throw an exception, processing continues in an exception handler.
* Place the statements that can cause an exception inside a try block, and the handler inside a catch clause.
* Checked exceptions are due to external circumstances that the programmer cannot prevent. The compiler checks that your program handles these exceptions.
* Add a throws clause to a method that can throw a checked exception.
* Once a try block is entered, the statements in a finally clause are guaranteed to be executed, whether or not an exception is thrown.
* Throw an exception as soon as a problem is detected. Catch it only when the problem can be handled.

**Use exception handling in a program that processes input.**

* When designing a program, ask yourself what kinds of exceptions can occur.
* For each exception, you need to decide which part of your program can competently handle it.

Arrays and Array Lists

**Use arrays for collecting values.**

* An array collects a sequence of values of the same type.
* Individual elements in an array are accessed by an integer index i, using the notation *array*[i].
* An array element can be used like any variable.
* An array index must be at least zero and less than the size of the array.
* A bounds error, which occurs if you supply an invalid array index, can cause your program to terminate.
* Use the expression *array*.length to find the number of elements in an array.
* An array reference specifies the location of an array. Copying the reference yields a second reference to the same array.
* With a partially filled array, keep a companion variable for the current size.

**Know when to use the enhanced for loop.**

* You can use the enhanced for loop to visit all elements of an array.
* Use the enhanced for loop if you do not need the index values in the loop body.

**Know and use common array algorithms.**

* When separating elements, don't place a separator before the first element.
* A linear search inspects elements in sequence until a match is found.
* Before inserting an element, move elements to the end of the array *starting with the last one*.
* Use a temporary variable when swapping two elements.
* Use the Arrays.copyOf method to copy the elements of an array into a new array.

**Implement methods that process arrays.**

* Arrays can occur as method arguments and return values.

**Combine and adapt algorithms for solving a programming problem.**

* By combining fundamental algorithms, you can solve complex programming tasks.
* You should be familiar with the implementation of fundamental algorithms so that you can adapt them.

**Discover algorithms by manipulating physical objects.**

* Use a sequence of coins, playing cards, or toys to visualize an array of values.
* You can use paper clips as position markers or counters.

**Use two-dimensional arrays for data that is arranged in rows and columns.**

* Use a two-dimensional array to store tabular data.
* Individual elements in a two-dimensional array are accessed by using two index values, *array*[i][j].

**Use array lists for managing collections whose size can change.**

* An array list stores a sequence of values whose size can change.
* The ArrayList class is a generic class: ArrayList<*Type*> collects elements of the specified type.
* Use the size method to obtain the current size of an array list.
* Use the get and set methods to access an array list element at a given index.
* Use the add and remove methods to add and remove array list elements.
* To collect numbers in array lists, you must use wrapper classes.

Objects and Classes

**Understand the concepts of classes, objects, and encapsulation.**

* A class describes a set of objects with the same behavior.
* Every class has a public interface: a collection of methods through which the objects of the class can be manipulated.
* Encapsulation is the act of providing a public interface and hiding the implementation details.
* Encapsulation enables changes in the implementation without affecting users of a class.

**Understand instance variables and method implementations of a simple class.**

* An object's instance variables store the data required for executing its methods.
* Each object of a class has its own set of instance variables.
* An instance method can access the instance variables of the object on which it acts.
* A private instance variable can only be accessed by the methods of its own class.

**Write method headers that describe the public interface of a class.**

* You can use method headers and method comments to specify the public interface of a class.
* A mutator method changes the object on which it operates.
* An accessor method does not change the object on which it operates.

**Choose an appropriate data representation for a class.**

* For each accessor method, an object must either store or compute the result.
* Commonly, there is more than one way of representing the data of an object, and you must make a choice.
* Be sure that your data representation supports method calls in any order.

**Provide the implementation of instance methods for a class.**

* The object on which a method is applied is the implicit parameter.
* Explicit parameters of a method are listed in the method declaration.

**Design and implement constructors.**

* A constructor initializes the instance variables of an object.
* A constructor is invoked when an object is created with the new operator.
* The name of a constructor is the same as the class name.
* A class can have multiple constructors.
* The compiler picks the constructor that matches the construction arguments.
* By default, numbers are initialized as 0, Booleans as false, and object references as null.
* If you do not provide a constructor, a constructor with no arguments is generated.

**Write tests that verify that a class works correctly.**

* A unit test verifies that a class works correctly in isolation, outside a complete program.
* To test a class, use an environment for interactive testing, or write a tester class to execute test instructions.
* Determining the expected result in advance is an important part of testing.

**Use the technique of object tracing for visualizing object behavior.**

* Write the methods on the front of a card, and the instance variables on the back.
* Update the values of the instance variables when a mutator method is called.

**Use patterns to design the data representation of a class.**

* An instance variable for the total is updated in methods that increase or decrease the total amount.
* A counter that counts events is incremented in methods that correspond to the events.
* An object can collect other objects in an array or array list.
* An object property can be accessed with a getter method and changed with a setter method.
* If your object can have one of several states that affect the behavior, supply an instance variable for the current state.

|  |  |
| --- | --- |
| |  | | --- | |  | |

* To model a moving object, you need to store and update its position.

**Describe the behavior of object references.**

* An object reference specifies the location of an object.
* Multiple object variables can contain references to the same object.
* Primitive type variables store values. Object variables store references.
* When copying an object reference, you have two references to the same object.
* The null reference refers to no object.
* In a method, the this reference refers to the implicit parameter.

**Understand the behavior of static variables and methods.**

* A static variable belongs to the class, not to any object of the class.
* A static method is not invoked on an object.

Inheritance and Interfaces

**Explain the notions of inheritance, superclass, and subclass.**

* A subclass inherits data and behavior from a superclass.
* You can always use a subclass object in place of a superclass object.

**Implement subclasses in Java.**

* A subclass inherits all methods that it does not override.
* A subclass can override a superclass method by providing a new implementation.
* The extends reserved word indicates that a class inherits from a superclass.

**Implement methods that override methods from a superclass.**

* An overriding method can extend or replace the functionality of the superclass method.
* Use the reserved word super to call a superclass method.
* Unless specified otherwise, the subclass constructor calls the superclass constructor with no arguments.
* To call a superclass constructor, use the super reserved word in the first statement of the subclass constructor.
* The constructor of a subclass can pass arguments to a superclass constructor, using the reserved word super.

**Use polymorphism for processing objects of related types.**

* A subclass reference can be used when a superclass reference is expected.
* Polymorphism (“having multiple shapes”) allows us to manipulate objects that share a set of tasks, even though the tasks are executed in different ways.
* An abstract method is a method whose implementation is not specified.
* An abstract class is a class that cannot be instantiated.

**Use the toString method and instanceof operator with objects.**

* Override the toString method to yield a string that describes the object's state.
* The equals method checks whether two objects have the same contents.
* If you know that an object belongs to a given class, use a cast to convert the type.
* The instanceof operator tests whether an object belongs to a particular type.

**Use interface types for algorithms that process objects of different classes.**

* A Java interface type contains the return types, names, and parameter variables of a set of methods.
* Unlike a class, an interface type provides no implementation.
* By using an interface type for a parameter variable, a method can accept objects from many classes.
* The implements reserved word indicates which interfaces a class implements.
* Implement the Comparable interface so that objects of your class can be compared, for example, in a sort method.

Decisions

**Use the if statement to implement a decision.**

* The if statement allows a program to carry out different actions depending on the nature of the data to be processed.

**Implement comparisons of numbers and objects.**

* Use relational operators (< <= > >= = !=) to compare numbers.
* Do not use the = = operator to compare strings. Use the equals method instead.
* The compareTo method compares strings in lexicographic order.

**Implement complex decisions that require multiple if statements.**

* Multiple if statements can be combined to evaluate complex decisions.
* When using multiple if statements, test general conditions after more specific conditions.

**Implement decisions whose branches require further decisions.**

* When a decision statement is contained inside the branch of another decision statement, the statements are nested.
* Nested decisions are required for problems that have two levels of decision making.

**Draw flowcharts for visualizing the control flow of a program.**

* Flow charts are made up of elements for tasks, input/output, and decisions.
* Each branch of a decision can contain tasks and further decisions.
* Never point an arrow inside another branch.

**Design test cases for your programs.**

* Each branch of your program should be covered by a test case.
* It is a good idea to design test cases before implementing a program.
* Logging messages can be deactivated when testing is complete.

**Use the Boolean data type to store and combine conditions that can be true or false.**

* The Boolean type boolean has two values, false and true.
* Java has two Boolean operators that combine conditions: && (and) and || (or).
* To invert a condition, use the ! (not) operator.
* The && and || operators are computed using short-circuit evaluation: As soon as the truth value is determined, no further conditions are evaluated.
* De Morgan's law tells you how to negate && and || conditions.

**Apply if statements to detect whether user input is valid.**

* Call the hasNextInt or hasNextDouble method to ensure that the next input is a number.

Loops

**Explain the flow of execution in a loop.**

* A loop executes instructions repeatedly while a condition is true.
* An off-by-one error is a common error when programming loops. Think through simple test cases to avoid this type of error.

**Use the technique of hand-tracing to analyze the behavior of a program.**

* Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.
* Hand-tracing can help you understand how an unfamiliar algorithm works.
* Hand-tracing can show errors in code or pseudocode.

**Use for loops for implementing count-controlled loops.**

* The for loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.

**Choose between the while loop and the do loop.**

* The do loop is appropriate when the loop body must be executed at least once.

**Implement loops that read sequences of input data.**

* A sentinel value denotes the end of a data set, but it is not part of the data.
* You can use a Boolean variable to control a loop. Set the variable to true before entering the loop, then set it to false to leave the loop.
* Use input redirection to read input from a file. Use output redirection to capture program output in a file.

**Use the technique of storyboarding for planning user interactions.**

* A storyboard consists of annotated sketches for each step in an action sequence.
* Developing a storyboard helps you understand the inputs and outputs that are required for a program.

**Know the most common loop algorithms.**

* To compute an average, keep a total and a count of all values.
* To count values that fulfill a condition, check all values and increment a counter for each match.
* If your goal is to find a match, exit the loop when the match is found.
* To find the largest value, update the largest value seen so far whenever you see a larger one.
* To compare adjacent inputs, store the preceding input in a variable.

**Use nested loops to implement multiple levels of iteration.**

* When the body of a loop contains another loop, the loops are nested. A typical use of nested loops is printing a table with rows and columns.

**Apply loops to the implementation of simulations.**

* In a simulation, you use the computer to simulate an activity.
* You can introduce randomness by calling the random number generator.