Supplement I.A: Glossary

For Introduction to Java Programming and Data Structures
By Y. Daniel Liang

Chapter 1

- **.class file** The output of the Java compiler. A .class file contains the byte code for the class.
- **.java file** The source code of a Java program. It may contain one or more Java classes and interfaces. A .java file can be created using a text editor or a Java IDE such as NetBeans, Eclipse, and JBuilder.
- **Application Program Interface** (**API**) A set of classes and interfaces that can be used to develop Java programs.
- **assembler** A program that translates assembly-language programs into machine code.
- **assembly language** A low-level programming language in which a mnemonic is used to represent each of the machine language instructions.
- **binary numbers** Numbers consists of 0's and 1's.
- **bit** A binary digit 0 and 1.
- **block** A sequence of statements enclosed in braces ({}).
- **block comment** Enclosed between /* and */ on one or several lines in the source code.
- **bus** A system that connects all the components of the computers together.
- **byte** A unit of storage. Each byte consists of 8 bits. The size of hard disk and memory is measured in bytes. A megabyte is roughly a million bytes.
- **bytecode** The result of compiling Java source code. Bytecode is machine-independent and can run on any machine that has a Java running environment.
- **Bytecode verifier** A program in the JVM that checks the validity of the bytecode and ensure that the bytecode does not violate Java's security restrictions.
- **cable modem** Uses the TV cable line maintained by the cable company. A cable modem is as fast as a DSL.
- **central processing unit** (**CPU**) A small silicon semiconductor chip with millions of transistors that executes instructions.
- **class loader** When executing a Java program, the JVM first loads the bytecode of the class to memory using a program called the *class loader*. If your program uses other classes, the class loader dynamically loads them just before they are needed.
- **comment** Comments document what a program is and how it is constructed. They are not programming statements and are ignored by the compiler. In Java, comments are preceded by two slashes (//) in a line or enclosed between /* and */ in multiple lines.
- **compiler** A software program that translates source code (e.g., Java source code) into a machine language program.

- **console** Refers to the input and output device of a computer.
- **dot pitch** The amount of space between pixels. The smaller the dot pitch, the better the display.
- **DSL** (**digital subscriber line**) Uses a phone line and can transfer data in a speed 20 times faster than a regular modem.
- **encoding scheme** is a set of rules that govern how a computer translates characters into data the computer can actually work with.
- **hardware** The physical aspect of the computer that can be seen.
- **hexadecimal numbers** Numbers with radix 16.
- **high-level language** Same as high-level programming language.
- **high-level programming language** Are English-like and easy to learn and program.
- **Integrated Development Environment** (**IDE**) Software that helps programmers write code efficiently. IDE tools integrate editing, compiling, building, debugging, and online help in one graphical user interface.
- **interpreter** Software for interpreting and running Java bytecode.
- **library** Same as Application Program Interface.
- **java command** The command to invoke the interpreter to run a Java program from the command line.
- **javac command** The command to invoke the compiler to compile a Java source code program from the command line.
- **Java Development Toolkit** (**JDK**) Defines the Java API and contains a set of command-line utilities, such as javac (compiler) and java (interpreter). With Java 2, Sun renamed JDK 1.5 to Java 2 SDK v 1.5. SDK stands for Software Development Toolkit.
- **Java Virtual Machine** (**JVM**) A machine that run Java byte-code. It is called virtual because it is usually implemented in software rather than in hardware.
- **Just-in-Time compiler** Capable of compiling each bytecode once, and then reinvoking the compiled code repeatedly when the bytecode is executed.
- **keyword** (or **reserved word**) A word defined as part of Java language, which have a specific meaning to the compiler and cannot be used for other purposes in the program.
- **line comment** comments preceded by two slashes (//).
- **logic error** (//).occur when a program does not perform the way it was intended to.
- **low-level language** refers to assembly language.
- **machine language** Is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions.
- **main class** A class that contains a main method.
- **main method** is the starting method for executing a program.
- **memory** Stores data and program instructions for CPU to execute.
- **modem** A regular modem uses a phone line and can transfer data in a speed up to 56,000 bps (bits per second).

- **motherboard** is a circuit case that connects all of the parts of a computer together.
- **network interface card** (**NIC**) A device to connect a computer to a local area network (LAN). The LAN is commonly used in business, universities, and government organizations. A typical type of NIC, called 10BaseT, can transfer data at 10 Mbps.
- **operating system** (**OS**) A program that manages and controls a computer's activities (e.g., Windows, Linux, Solaris).
- **pixel** Tiny dots that form an image on the screen.
- **program** Same as software.
- **programming** is the process for writing a program.
- **resolution** Specifies the number of pixels per square inch. The higher the resolution, the sharper and clearer the image is.
- **runtime error** are errors that cause a program to terminate abnormally.
- **software** The invisible instructions that control the hardware and make it work.
- **source code** A program written in a programming language such as Java.
- **source program** Same as source code.
- **source file** A file that stores the source code.
- **storage devices** The permanent storage for data and programs. Memory is volatile, because information is lost when the power is off. Program and data are stored on secondary storage and moved to memory when the computer actually uses them.
- **statement** A unit of code that represents an action or a sequence of actions.
- **statement terminator** The semicolon is the statement terminator symbol in Java.
- **syntax error** are errors that reported by the compiler.

Chapter 2

- **algorithm** Statements that describe how a problem is solved in terms of the actions to be executed, and specifies the order in which these actions should be executed. Algorithms can help the programmer plan a program before writing it in a programming language.
- **assignment operator** (=) Assigns a value to a variable.
- **assignment statement** A simple statement that assigns a value to a variable using an assignment operator (=). When a value is assigned to a variable, it replaces the previous value of the variable, which is destroyed.
- **backslash** (\) -- A character that precedes another character to denote the following character has a special meaning. For example, '\t' denote a tab character. The backslash character is also used to denote a Unicode character like '\u00FF'.
- **byte type** A primitive data type that represents an integer in a byte. The range a byte value is from -2^7 (-128) to 2^7-1 (127).
- **casting** The process of converting a primitive data type value into another primitive type.
- **constant** A variable declared final in Java. A local constant is a constant declared inside a method.
- **data type** Used to define variables to indicate what kind of value the variable can hold.
- **debugger** A program that facilitates debugging. It enables the program to be executed one statement at a time and enables the contents of the variable to be examined during execution.
- **debugging** The process of finding and fixing errors in a program.
- **declare variable** defines a variable with a data type.
- **declaration** Defines variables, methods, and classes in a program.
- **decrement operator** (--) Subtracts one from a numeric variable or a char variable.
- **double type** A primitive data type to represent double precision floating-point numbers with 14 to 15 significant digits of accuracy.
- **final keyword** A modifier that specifies a constant.
- **float type** A primitive data type to represent single precision floating-point numbers with 6 to 7 significant digits of accuracy. The double type is used to represent double precisions with 14 to 15 significant digits of accuracy.
- **floating-point number** A number that includes a fractional part.
- **expression** Represents a computation involving values, variables, and operators, which evaluates to a value.
- **expression statement** If an expression is used as a statement, it is called an expression statement.
- **identifier** A name of a variable, method, class, interface, or package.
- **increment operator** (++) Adds one to a numeric variable or a char variable.

- **incremental code and testing** A programming methodology that develop and test program incrementally. This approach is efficient and productive. It helps eliminate and isolate errors.
- **indentation** The use of tabs and spaces to indent the source code to make it easy to read and understand.
- **int type** A primitive data type to represent an integer in the range from -2^{31} (-2147483648) to $2^{31}-1$ (2147483647).
- **IPO** stands for Input, Process, and Output.
- **literal** A constant value that appears directly in the program. A literal may be numeric, character, boolean, or null for object type.
- **logic error** An error that causes the program to produce incorrect result.
- **long type** A primitive data type to represent an integer in the range from -2^{63} to $2^{63}-1$.
- **narrowing** (**of types**) Casting a variable of a type with a larger range to a variable of a type with a smaller range.
- **operands** are the values operated by an operator.
- **operator** Operations for primitive data type values. Examples of operators are +, -, *, /, and %.
- **overflow** When a variable is assigned a value that is too large (in size) to be stored, it causes overflow.
- **postdecrement** refers to the syntax such as x-- where the -- operator is place after a variable.
- **postincrement** refers to the syntax such as x++ where the ++ operator is place after a variable.
- **predecrement** refers to the syntax such as --x where the -- operator is place before a variable.
- **preincrement** refers to the syntax such as ++x where the ++ operator is place before a variable.
- **primitive data type** The primitive data types are byte, short, int, long, float, double, boolean, and char.
- **pseudocode** describes the program logic using natural language mixed with some programming code.
- **requirement specification** is a formal process that seeks to understand the problem that the software will address and to document in detail what the software system needs to do.
- **scope of a variable** is the part of the program where the variable can be referenced.
- **short type** A primitive data type that represents an integer in the range from -2^{15} (-32768) to $2^{15}-1$ (32767).
- **specific import** specifies a single class in the import statement. For example, import java.util.Scanner imports Scanner from package java.util.
- **system analysis** seeks to analyze the data flow and to identify the system's input and output. When you do analysis, it helps to identify what the output is first, and then figure out what input data you need in order to produce the output.

- **system design** is to design a process for obtaining the output from the input. This phase involves the use of many levels of abstraction to break down the problem into manageable components and design strategies for implementing each component.
- **underflow** When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Java approximates it to zero, so normally you don't need to be concerned about underflow.
- **Unix epoch** January 1, 1970 GMT is known as the *Unix epoch* because 1970 was the year when the Unix operating system was formally introduced.
- **variable** Variables are used to store data and computational results in the program.
- **widening** (**of types**) Casting a variable of a type with a smaller range to a variable of a type with a larger range.
- **wildcard import** imports all the classes in a package. For example, import java.util.* imports all classes from package java.util.

Chapter 3

- **boolean expression** An expression that evaluates to a Boolean value.
- **boolean value** true or false.
- **boolean data type** A primitive data type for Boolean values (true or false).
- **boolean type** Same as boolean type.
- **break statement** Break out of the switch statement.
- **conditional operator** The symbols ? and : appear together in a conditional expression: booleanExpression ? expression1 : expression2;
- **dangling else ambiguity** describes a common mistake where an else clause is mismatch to an if clause.
- **debugging** is to find errors in a program.
- **fall-through behavior** In a switch statement, if Once a case is matched, the statements starting from the matched case are executed until a break statement or the end of the switch statement is reached. This phenomenon is referred to as the fall-through behavior.
- **flowchart** is a diagram that describes an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows.
- **lazy operator** Same as short-circuit operator.
- **operand evaluation order** Defines the order in which individual operands are evaluated.
- **operator associativity** Defines the order in which operators will be evaluated in an expression if the operators has the same precedence order.
- **operator precedence** Defines the order in which operators will be evaluated in an expression.
- **selection statement** A statement that uses if or switch statement to control the execution of the program.
- **short-circuit evaluation** Evaluation that stops when the result of the expression has been determined, even if not all the operands are evaluated. The evaluation involving && or || are examples of short-circuit evaluation.
- **short-circuit operator** Operators such as && and || that perform short-circuit evaluation.

Chapter 4

- **ASCII** code American Standard Character Encoding Scheme.
- **backslash** (\) A character that precedes another character to denote the following character has a special meaning. For example, '\t' denote a tab character.
- **char type** A primitive data type that represents a Unicode character.
- **encoding** Representing a character using a binary code.
- **format specifier** A format specifier specifies how an item should be displayed. An item may be a numeric value, a character, a Boolean value, or a string.
- **empty string** A string containing no characters.
- **escape character** The backslash \ is called an escape character.
- **escape sequence** A special notation, consists of a backslash (\) followed by a character or a combination of digits.
- **instance method** A method that must be invoked from a specific object.
- **static method** A method invoked from a class name.
- **supplementary Unicode** The original Unicode is 16-bit. Those characters that go beyond the original 16-bit limit are called supplementary characters.
- **Unicode** A code system for international characters managed by the Unicode Consortium. Java supports Unicode.
- **whitespace characters** The characters ' ', \t, \f, \r, or \n are known as whitespace characters.

Chapter 5

- **break statement** Break out of the current loop.
- **control-variable** is a variable commonly used to control the execution of a loop.
- **continue statement** Break out of the current iteration.
- **counter-controlled loop** is a type of loop that uses a control variable to count the number of iterations in a loop.
- **do-while loop** A loop construct that begins with the keyword do.
- **for loop** A loop construct that begins with the keyword for.
- **infinite loop** A loop that runs indefinitely due to a bug in the loop.
- **input redirection** Obtain the input from a file instead of from the console.
- **iteration** One time execution of the loop body.
- **labeled break statement** Break out of the specified labeled loop.
- **labeled continue statement** Break out of the current iteration of the labeled loop.
- **loop** A structure that control repeated executions of a block of statements.
- **loop-continuation-condition** A Boolean expression that controls the execution of the body. After each iteration, the loop-continuation-condition is reevaluated. If the condition is true, the execution of the loop body is repeated. If the condition is false, the loop terminates.
- **loop body** The part of the loop that contains the statements to be repeated.
- **nested loop** Consists of an outer loop and one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered, and all required iterations are performed.
- **off-by-one error** A common in the loop because the loop is executed one more or one less time than it should have been.
- **output redirection** Write the output to a file instead of to the console.
- **posttest loop** The loop continuation condition is checked after the loop body is executed.
- **pretest loop** The loop continuation condition is checked before the loop body is executed.
- **while loop** A loop construct that begins with the keyword while.
- **sentinel value** A special input value that signifies the end of the input.

Chapter 6

- **actual parameter** (i.e., argument) The variables or data to substitute formal parameters when invoking a method.
- **ambiguous invocation** There are two or more possible methods to match an invocation of a method, neither is more specific than the other(s). Therefore, the invocation is ambiguous.
- **argument** Same as actual parameter
- **bottom-up implementation** Implementing and testing methods incrementally from the bottom along the chain of the methods.
- **divide and conquer** The concept of method abstraction can be applied to the process of developing programs. When writing a large program, you can use the "divide and conquer" strategy to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.
- **formal parameter (i.e., parameter)** The variables defined in the method signature.
- **information hiding** A software engineering concept for hiding the detail implementation of a method for the client.
- **method** A collection of statements grouped together to perform an operation. See class method; instance method.
- **method abstraction** A technique in software development that hides detailed implementation. Method abstraction is defined as separating the use of a method from its implementation. The client can use a method without knowing how it is implemented. If you decide to change the implementation, the client program will not be affected.
- **method overloading** Method overloading means that you can define methods with the same name in a class as long as there is enough difference in their parameter profiles.
- **method signature** The combination of the name of a method and the list of its parameters.
- **modifier** A Java keyword that specifies the properties of data, methods, and classes and how they can be used. Examples of modifiers are public, private, and static.
- **package** A mechanism for organizing classes.
- **parameter** A variable that is defined in the method header.
- **pass-by-value** (i.e., call-by-value) A term used when a copy of the value of the argument is passed to the method. For a parameter of a primitive type, the actual value is passed; for a parameter of a reference type, the reference for the object is passed.
- **return type** The data type for the return value in a method.
- **return value** A value returned from a method using the return statement.
- **scope of variable** The portion of the program where the variable can be accessed.
- **stepwise refinement** When writing a large program, you can use the "divide and

conquer” strategy, also known as stepwise refinement, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

- **stub** A simple, but not a complete version of the method. The use of stubs enables you to test invoking the method from a caller.

Chapter 7

- **anonymous array** An array created without an explicit reference.
- **array** A data structure for storing a collection of data of the same type.
- **array initializer** A short hand notation to create and initialize an array.
- **binary search** An efficient method to search a key in an array. Binary search first compares the key with the element in the middle of the array and reduces the search range by half. For binary search to work, the array must be pre-sorted.
- **garbage collection** A JVM feature that automatically detects and reclaims the space occupied by unreferenced objects.
- **index** An integer value used to specify the position of an element in the array. The array index is an int value starting with 0 for the first element, 1 for the second, and so on.
- **indexed variable** A variable that is represented using an array reference along with an index to refer to an individual element in the array.
- **linear search** A method to search an element in an array. Linear search compares the key with the element in the array sequentially.
- **off-by-one error** Programmers often mistakenly reference the first element in an array with index **1**, but it should be **0**. This is called the `off-by-one error`.
- **selection sort** An approach to sort array. It finds the largest number in the list and places it last. It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number.

Chapter 9

- **accessor method** (**getter**) The method for retrieving a private field in an object.
- **anonymous object** An object created without assigned to a reference variable.
- **action** same as behavior.
- **attribute** A variable that stores a value for an object.
- **behavior** A method that can be called by an object.
- **class** An encapsulated collection of data and methods that operate on data. A class may be instantiated to create an object that is an instance of the class.
- **class's variable** refers to the variables defined in the class.
- **client** refers the program that uses a class. The program is known as a client of the class.
- **constructor** A special method for initializing objects when creating objects using the new operator. The constructor has exactly the same name as its defining class. Constructors can be overloaded, making it easier to construct objects with different initial data values.
- **data field** Same as attribute.
- **data field encapsulation** To prevent direct modifications of properties through the object reference, you can declare the field private, using the private modifier. Data field encapsulation makes the class easy to maintain.
- **default constructor** If a class does not define any constructors explicitly, a no-arg constructor with empty body is assumed. This is called a default constructor.
- **dot operator** (**.**) An operator used to access members of an object. If the member is static, it can be accessed through the class name using the dot operator.
- **getter** Same as accessor method.
- **instance** An object of a class.
- **instance method** A nonstatic method in a class. Instance methods belong to instances and can only be invoked by them.
- **instance variable** A nonstatic data member of a class. An instance variable belongs to an instance of the class.
- **instantiation** The process of creating an object of a class.
- **immutable class** A class is immutable if it contains all private data fields and no mutator methods and no accessor methods that would return a reference to a mutable data field object.
- **immutable object** An object of immutable class.
- **mutator method** (**setter**) A method that changes the value of a private field in an object.
- **null** A literal of a reference variable that does not reference any concrete object.
- **no-arg constructor** A constructor without arguments.
- **object** can be created from a class using the class's constructor.
- **object-oriented programming (OOP)** An approach to programming that involves organizing objects and their behavior into classes of reusable components.
- **Unified Modeling Language** (**UML**) A graphical notation for describing

classes and their relationships.

- **package-private** (or **package-access**) If public or private is not used, then by default the classes, methods, and data are accessible by any class in the same package. This is known as package-private or package-access.
- **private** A modifier for members of a class. A private member can only be referenced inside the class.
- **private constructor** A private constructor prevents constructing objects from the class.
- **property** Same as attribute.
- **public** A modifier for classes, data, and methods that can be accessed by all programs.
- **public class** The public visibility modifier is used for the class.
- **reference** A value that references an object.
- **reference type** A data type that is a class or an interface.
- **reference variable** A variable of a class type.
- **state** The state of an object is described by attributes of the object.
- **setter** Same as mutator method.
- **static method** A method that can be invoked without creating an instance of the class. To define static methods, put the modifier static in the method declaration.
- **static variable** A data member declared using the static modifier. A static variable is shared by all instances of that class. Static variables are used to communicate between different objects of the same class and to handle global states among these objects.
- **this keyword** Refers to the object itself.
- **Unified Modeling Language** A graphic notation for describing classes and objects.

Chapter 10

- **abstract data type** A class is also known as an abstract data type.
- **aggregation** A special form of association that represents an ownership relationship between two classes.
- **boxing** Converting a primitive value to a wrapper object is called boxing.
- **class abstraction** A technique in software development that hides detailed implementation. Class abstraction hides the implementation of the class from the client, if you decide to change the implementation, the client program will not be affected.
- **class encapsulation** Combining of methods and data into a single data structure.
- **class's contract** Refers to the collection of methods and fields that are accessible from outside a class, together with the description of how these members are expected to behave.
- **class's variable** Instance and static variables in a class are referred to as the class's variables or data fields.
- **composition** An object consists of other objects. This is called composition.
- **has-a relationship** composition models a has-a relationship.
- **multiplicity** Each class involved in a relationship may specify a multiplicity. A multiplicity could be a number or an interval that specifies how many of the class's objects are involved in the relationship.
- **stack** A stack is a data structure that holds objects in a last-in first-out fashion.
- **unboxing** Converting a primitive value to a wrapper object is called boxing. The reverse conversion is called unboxing.

Chapter 11

- **actual type** The actual type of the variable is the actual class for the object referenced by the variable.
- **array list** A data structure for storing a list of array. The list size can grow and shrink.
- **casting objects** Converting an object of one object type into another object type. The contents of the object are not changed.
- **constructor chaining** Constructing an instance of a class invokes all the constructor, chaining superclasses along the inheritance chain.
- **declared type** A data type that is used to declare a reference variable. This type is called the declared type for the variable.
- **dynamic binding** A method may be defined in a superclass, but is overridden in a subclass. Which implementation of the method is used on a particular call will be determined dynamically by the JVM at runtime. This capability is known as dynamic binding.
- **final** A modifier for classes, data, methods, and local variables. A final class cannot be extended, a final data or local variable is a constant, and a final method cannot be overridden in a subclass.
- **generic programming** Allows methods to be used generically for a wide range of object arguments through polymorphism.
- **inheritance** Defining a new class by extending an existing class.
- **instanceof** An operator that checks whether an object is an instance of a class.
- **is-a relationship** Same as inheritance.
- **method overriding** Same as overriding.
- **multiple inheritance** means that a class can extend multiple superclasses.
- **override** Implement the method in a subclass that is declared in a superclass.
- **polymorphism** Refers to the feature that an object of a subclass can be used by any code designed to work with an object of its superclass.
- **protected** A modifier for members of a class. A protected member of a class can be used in the class in which it is declared or any subclass derived from that class.
- **single inheritance** means that a class can only extend one superclass.
- **subclass** (**child class** or **derived class**) A class that inherits from or extends a superclass.
- **subtype** Same as subclass.
- **superclass** (**parent** class or **extended class**) A class inherited from a subclass.
- **supertype** Same as superclass.
- **type inference** The concrete type is no longer required in the constructor thanks to a feature called type inference.

Chapter 12

- **absolute filename** An absolute file name (or full name) contains a file name with its complete path and drive letter.
- **chained exception** Throw new exceptions along with the original exception.
- **checked exception** Exceptions other than RuntimeException and Error.
- **declare exception** All checked exceptions thrown by the method must be explicitly declared in the method declaration so that the caller of the method is informed of the exception.
- **directory path** a path for a directory in the file system.
- **exception** An unexpected event indicating that a program has failed in some way. Exceptions are represented by exception objects in Java. Exceptions can be handled in a try-catch block.
- **exception propagation** refers to the mechanism for catching exceptions in a chain of method calls. The exception is propagated through a chain of methods.
- **relative filename** refers to a filename without the full path. Its path is implied in its working directory.
- **throw exception** A program that detects an error can create an instance of an appropriate exception type and throw it. This is known as *throwing an exception*.
- **unchecked exception** Instances of RuntimeException and Error.

Chapter 13

- **abstract class** When you are designing classes, a superclass should contain common features that are shared by subclasses. Sometimes the superclass is so abstract that it cannot have any specific instances. These classes are called abstract classes and are declared using the abstract modifier. Abstract classes are like regular classes with data and methods, but you cannot create instances of abstract classes using the new operator.
- **abstract method** A method signature without implementation. Its implementation is provided by its subclasses. An abstract method is denoted with an abstract modifier and must be contained in an abstract class. In a nonabstract subclass extended from an abstract class, all abstract methods must be implemented, even if they are not used in the subclass.
- **deep copy** When cloning an object, all its fields are cloned recursively.
- **interface** An interface is treated like a special class in Java. Each interface is compiled into a separate bytecode file, just like a regular class. You cannot create an instance for an interface. The structure of a Java interface is similar to that of an abstract class in that it can have data and methods. The data, however, must be constants, and the methods can have only declarations without implementation. Single inheritance is the Java restriction wherein a class can inherit from a single superclass. This restriction is eased by the use of an interface.
- **marker interface** An empty interface that is used to signify that all the classes implementing the interface share certain properties.
- **multiple inheritance** A class may extend multiple superclasses.
- **shallow copy** When cloning an object, all its fields are copied.
- **single inheritance** A class can extend only one superclass.
- **subinterface** An interface inherited from other interface.

Chapter 14

- **AWT** Abstract Window Toolkit (AWT) The set of components for developing simple graphics applications that was in use before the introduction of Swing components.
- **bidirectional binding** Two properties are bound together so that a change in one property is reflected in another object, and vice versa.
- **bindable object** When a target object is bound to a source object, the source object is called a bindable object or observable object.
- **JavaFX** JavaFX is a platform for GUI programming in Java. It incorporates modern GUI technologies to enable you to develop rich Internet applications.
- **binding object** JavaFX introduces a new concept called binding property that enables a target object to be bound to a source object. If the value in the source object changes, the target property is also changed automatically. The target object is simply called a binding object or a binding property.
- **binding property** Same as binding object.
- **node** A node is a visual component such as a shape, an image view, a UI control, or a pane.
- **observable object** Same as bindable object.
- **pane** A pane is a JavaFX node for grouping and organizing other nodes.
- **property getter method** A getter method for returning the property itself.
- **primary stage** A Stage object is a window. A Stage object called primary stage is automatically created by the JVM when the application is launched.
- **shape** A shape refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- **Swing** Swing is a platform for developing GUI programs in Java. It has been replaced by JavaFX.
- **value getter method** A getter method that returns the value in a property.
- **value setter method** A setter method that sets the value in a property.
- **UI control** A UI control refers to a label, button, check box, radio button, text field, text area, etc.
- **unidirectional binding** A property is bound to a source property. Change of the value in the source property is automatically reflected in the target. This is called a unidirectional binding.

Chapter 15

- **anonymous inner class** An inner class without a name.
- **event** A signal to the program that something has happened. Events are generated by external user actions, such as mouse movements, mouse button clicks, and keystrokes, or by the operating system, such as a timer. The program can choose to respond to an event or to ignore it.
- **event delegation** In Java event-driven programming, events are assigned to the listener object for processing. This is referred to as event delegation.
- **event-driven programming** is to write GUI programs that response to interactive events.
- **Event-handler interface** An interface that defines a method for handling events.
- **event handler** An object that handles the event.
- **event listener** Same as event handler
- **event listener interface** An interface implemented by the listener class to handle the specified events.
- **event object** Contains whatever properties are pertinent to the event.
- **event source object** (or **source object**) The object that generates the event.
- **functional interface** An interface that has only one method.
- **lambda expression** Lambda expressions can be viewed as an anonymous class with a concise syntax.
- **inner class** A class embedded in another class. Inner classes enable you to define small auxiliary objects and pass units of behavior, thus making programs simple and concise.
- **key code** The key codes are constants defined in KeyCode.
- **inner class** A class embedded in another class. Inner classes enable you to
- **observable object** is an instance of the Observable interface.
- **single abstract method interface** Same as functional interface.

Chapter 17

- **binary I/O** Binary I/O interprets data as raw binary values.
- **deserialization** The process of restoring an object that was previously serialized.
- **file pointer** A location marker in a random access file where data is read and written.
- **random access file** The file that can be both read and written in any order.
- **sequential access file** The file is read or written sequentially from beginning to end.
- **serialization** The process of writing an object to a stream.
- **stream** A stream is an object that facilitates input or output. For input, it is called an input stream. For output, it is called an output stream.
- **text I/O** Text I/O interprets data in sequences of characters.

Chapter 18

- **base case** A simple case where recursion stops.
- **direct recursion** A recursive method that invokes itself.
- **indirect recursion** Method A invokes B, B then invokes A.
- **infinite recursion** Recursion never stops.
- **recursive method** A method that invokes itself directly or indirectly.
- **recursive helper method** Sometimes the original method needs to be modified to receive additional parameters in order to be invoked recursively. A recursive helper method can be declared for this purpose.
- **stopping condition** Same as base case.
- **tail recursion** A recursive method is said to be tail recursive if there are no pending operations to be performed on return from a recursive call.

Chapter 19

- **actual concrete type** A concrete type that substitutes a generic type.
- **bounded generic type** A generic type with a bound (e.g., <E extends SomeClass>).
- **formal generic type** A generic type.
- **generic instantiation** The process that instantiates a generic type with a concrete type.
- **raw type** For backward compatibility, a generic class may be used without specifying a concrete class.
- unbounded wildcard **<?> type** A wildcard that represents any object type.
- bounded wildcard **<? extends E>** declares a type that is a subtype of generic type E.
- lower-bound wildcard **<? super E>** declares a type that is a supertype of generic type E.

Chapter 20

- **collection** An object that contains a set or a list of objects.
- **comparator** A collection of ordered elements with duplicates allowed.
- **convenience abstract class** A class that partially implements an interface.
- **data structure** is a collection of data organized in some fashion. The structure not only stores data but also supports operations for accessing and manipulating the data.
- **list** A collection of ordered elements with duplicates allowed.
- **lnked list** is a list that is implemented using linked nodes.
- **map** A collection of entries, each consists of a key and an object.
- **priority queue** In a priority queue, elements are assigned with priorities. When accessing elements, the element with the highest priority is removed first.
- **queue** A collection of entries, each consists of a key and an object.

Chapter 21

- **hash map** A map in which entries are stored in unpredictable order.
- **hash set** A set in which elements are stored in unpredictable order.
- **linked hash map** A map in which entries are stored in certain order (insertion order or access order).
- **linked hash set** A map in which elements are stored in certain order (insertion order or access order).
- **map** is a data structure that stores entries of key-value pairs.
- **read-only view** refers to read-only data structures.
- **set** A collection of nonduplicate elements.
- **tree map** A map in which the keys are sorted.
- **tree set** A set in which the elements are sorted.

Chapter 22

- **average-case analysis** An average-case analysis attempts to determine the average amount of time among all possible input of the same size.
- **backtracking approach** . The backtracking approach searches for a candidate solution incrementally, abandoning that option as soon as it determines that the candidate cannot possibly be a valid solution, and then looks for a new candidate.
- **best-case input** An input that results in the shortest execution time is called the best-case input.
- **best-time analysis** The analysis to find the best-case time is known as worst-time analysis.
- **big O notation** Comparing algorithms by examining their growth rates. This notation allows you to ignore constants and smaller terms while focusing on the dominating terms.
- **brute force** refers to an algorithmic approach that solves a problem in the simplest or most direct or obvious way.
- **constant time** The Big O notation estimates the execution time of an algorithm in relation to the input size. If the time is not related to the input size, the algorithm is said to take constant time with the notation $O(1)$.
- **convex hall** Given a set of points, a *convex hul* is the smallest convex polygon that encloses all these points.
- **divide-and-conquer approach** divides the problem into subproblems, solves the subproblems, then combines the solutions of the subproblems to obtain the solution for the entire problem.
- **dynamic programming approach** is the process of solving subproblems, then combining the solutions of the subproblems to obtain an overall solution. This naturally leads to a recursive solution. However, it would be inefficient to use recursion, because the subproblems overlap. The key idea behind dynamic programming is to solve each subproblem only once and store the results for subproblems for later use to avoid redundant computing of the subproblems.
- **exponential time** An algorithm with the $O(c^n)$ time complexity is called an exponential algorithm. As the input size increases, the time for the exponential algorithm grows exponentially. The exponential algorithms are not practical for large input size.
- **growth rate** measures how fast the time complexity of an algorithm grows as the input size grows.
- **logarithmic time** An algorithm with the $O(\log n)$ time complexity is called a logarithmic algorithm.
- **quadratic time** An algorithm with the $O(n^2)$ time complexity is called a quadratic algorithm.
- **space complexity** is the analysis on the space used for the algorithm.
- **time complexity** is the analysis on the time used for the algorithm.
- **worst-case input** An input that results in the longest execution time is called the

worst-case input.

- **worst-time analysis** The analysis to find the worst-case time is known as worst-time analysis.

Chapter 23

- **bubble sort** The bubble sort algorithm makes several passes through the array. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a bubble sort or sinking sort because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom.
- **bucket sort** The bucket sort algorithm works as follows. Assume the keys are in the range from 0 to N-1. We need N buckets labeled 0, 1, ..., and N-1. If an element's key is i, the element is put into the bucket i. Each bucket holds the elements with the same key value.
- **complete binary tree** A binary tree is *complete* if each of its levels is full, except that the last level may not be full and all the leaves on the last level are placed leftmost.
- **external sort** sort data stored in an external file.
- **heap** (also known as binary heap) A *binary heap* is a binary tree with the following properties: 1. Shape property: It is a complete binary tree. 2. Heap property: Each node is greater than or equal to any of its children.
- **heap sort** Heap sort uses a binary heap to sort an array.
- **height of a heap** The *height* of a non-empty heap is the length of longest path from the root to a leaf node. The height of a heap that contains a single element is 0.
- **insertion sort** An approach to sort array. Suppose that you want to sort a list in ascending order. The insertion-sort algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole list is sorted.
- **merge sort** The merge sort algorithm can be described recursively as follows: The algorithm divides the array into two halves and applies merge sort on each half recursively. After the two halves are sorted, merge them.
- **quick sort** Quick sort, developed by C. A. R. Hoare (1962), works as follows: The algorithm selects an element, called the pivot, in the array. Divide the array into two parts such that all the elements in the first part are less than or equal to the pivot and all the elements in the second part are greater than the pivot. Recursively apply the quick sort algorithm to the first part and then the second part.
- **radix sort** Radix sort is like bucket sort, but it is based on radix.

Chapter 25

- **binary search tree** A BST (with no duplicate elements) has the property that for every node in the tree, the value of any node in its left subtree is less than the value of the node, and the value of any node in its right subtree is greater than the value of the node.
- **binary tree** A binary tree is a hierarchical structure. It either is empty or consists of an element, called the *root*, and two distinct binary trees, called the *left subtree* and *right subtree*, either or both of which may be empty.
- **breath-first traversal** With breadth-first traversal, the nodes are visited level by level. First the root is visited, then all the children of the root from left to right, then the grandchildren of the root from left to right, and so on.
- **depth** The *depth* of a node is the length of the path from the root to the node.
- **depth-first traversal** Depth-first traversal is to visit the root, then recursively visit its left subtree and right subtree in an arbitrary order. The preorder traversal can be viewed as a special case of depth-first traversal, which recursively visit its left subtree and then its right subtree.
- **greedy algorithm** A greedy algorithm is often used in solving optimization problems. The algorithm makes the choice that is optimal locally in the hope that this choice will lead to a globally optimal solution.
- **heap** Heaps are a useful data structure for designing efficient sorting algorithms and priority queues. A *heap* is a binary tree with the following properties: (1) It is a complete binary tree. (2) Each node is greater than or equal to any of its children.
- **height** The height of a nonempty tree is the length of the path from the root node to its furthest leaf. The *height* of a tree that contains a single node is 0. Conventionally, the height of an empty tree is −1.
- **Huffman coding** Huffman coding compresses data by using fewer bits to encode characters that occur more frequently. The codes for the characters are constructed based on the occurrence of the characters in the text using a binary tree, called the Huffman coding tree.
- **inorder traversal** With inorder traversal, the left subtree of the current node is visited first, then the current node, and finally the right subtree of the current node. The inorder traversal displays all the nodes in a BST in increasing order.
- **leaf** A node without children is called a *leaf*.
- **length** The *length* of a path is the number of the edges in the path.
- **level** The set of all nodes at a given depth is sometimes called a *level* of the tree.
- **postorder traversal** With postorder traversal, the left subtree of the current node is visited first, then the right subtree of the current node, and finally the current node itself.
- **preorder traversal** With preorder traversal, the current node is visited first, then the left subtree of the current node, and finally the right subtree of the current node. Depth-first traversal is the same as preorder traversal.

- **sibling** *Siblings* are nodes that share the same parent node.
- **tree traversal** Tree traversal is the process of visiting each node in the tree exactly once.

Chapter 26

- **AVL tree** An AVL tree is a *well-balanced* binary tree. In an AVL tree, the difference between the heights of two subtrees for every node is `0` or `1`.

- **balance factor** The *balance factor* of a node is the height of its right subtree minus the height of its left subtree.

- **left-heavy** A node is considered *left-heavy* if its balance factor is `-1` or less.

- **LL rotation** An *LL imbalance* occurs at a node `A`, such that `A` has a balance factor of `-2` and a left child `B` with a balance factor of `-1` or `0`. This type of imbalance can be fixed by performing a single right rotation at `A`. This rotation is called an LL rotation.

- **LR rotation** An *LR imbalance* occurs at a node `A`, such that `A` has a balance factor of `-2` and a left child `B` with a balance factor of `+1`. Assume `B`'s right child is `C`. This type of imbalance can be fixed by performing a double rotation (first a single left rotation at `B` and then a single right rotation at `A`). This rotation is called an LR rotation.

- **perfectly-balance tree** is a complete binary tree.

- **right-heavy** A node is considered *right-heavy* if its balance factor is `+1` or greater.

- **RL rotation** An *RL imbalance* occurs at a node `A`, such that `A` has a balance factor of `+2` and a right child `B` with a balance factor of `-1`. Assume `B`'s left child is `C`. This type of imbalance can be fixed by performing a double rotation (first a single right rotation at `B` and then a single left rotation at `A`). This rotation is called an RL rotation.

- **rotation** is a process for rebalance a tree into an AVL tree.

- **RR rotation** An *RR imbalance* occurs at a node `A`, such that `A` has a balance factor of `+2` and a right child `B` with a balance factor of `+1` or `0`. This type of imbalance can be fixed by performing a single left rotation at `A`. This rotation is called an RR rotation.

- **well-balance tree** A tree is said to be well-balance if the height of the subtrees for each node is about the same.

Chapter 27

- **associative array** An alternative term for map.

- **cluster** Linear probing tends to cause groups of consecutive cells in the hash table to be occupied. Each group is called a *cluster*.

- **dictionary** Same as map.

- **double hashing** Double hashing uses a secondary hash function *h'*(*key*) on the keys to determine the increments to avoid the clustering problem.

- **hash code** A hash code is an integer value derived from an object.

- **hash function** The function that maps a key to an index in the hash table is called a hash function.

- **hash map** is a map that is implemented using a hash table.

- **hash set** is a set that is implemented using a hash table.
.
- **hash table** is essentially an array that stores the elements whose keys are mapped to an integers as the indexes of the array.

- **linear probing** is a technique for handling collision. When a collision occurs during the insertion of an entry to a hash table, *linear probing* finds the next available location sequentially.

- **load factor** measures how full a hash table is. It is the ratio of the number of elements to the size of the hash table, that is, $\lambda = \frac{n}{N}$, where *n* denotes the number of elements and *N* the number of locations in the hash table.

- **open addressing** is the process of finding an open location in the hash table in the event of collision.

- **perfect hash function** is a function that maps each search key to a different index in the hash table.

- **polynomial hash code** is a polynomial function used for obtaining a hash code.

- **quadratic probing** can avoid the clustering problem that can occur in linear probing. Linear probing looks at the consecutive cells beginning at index *k*. Quadratic probing, on the other hand, looks at the cells at indices k + a quadratic

number.

- **rehashing** The load factor measures how full a hash table is. If the load factor is exceeded, increase the hash-table size and reload the entries into a new larger hash table. This is called rehashing.

- **separate chaining** is a hashing scheme that places all entries with the same hash index into the same location, rather than finding new locations. Each location in the separate chaining scheme is called a *bucket*. A bucket is a container that holds multiple entries.

Chapter 28

- **adjacency list** To represent edges using adjacency lists, define an array of linked lists. The array has *n* entries. Each entry represents a vertex. The linked list for vertex *i* contains all the vertices *j* such that there is an edge from vertex *i* to vertex *j*.
- **adjacent vertices** Two vertices in a graph are said to be *adjacent* if they are connected by the same edge.
- **adjacency matrix** representing edges using a matrix.
- **breadth-first search** first visits a vertex, then all its adjacent vertices, then all the vertices adjacent to those vertices, and so on. To ensure that each vertex is visited only once, skip a vertex if it has already been visited.
- **complete graph** A complete graph is the one in which every two pairs of vertices are connected.
- **cycle** A *cycle* is a closed path that starts from a vertex and ends at the same vertex.
- **degree** The *degree* of a vertex is the number of edges incident to it.
- **depth-first search** first visits the root, then recursively visits the subtrees of the root.
- **directed graph** In a directed graph, each edge has a direction, which indicates that you can move from one vertex to the other through the edge.
- **graph** A *graph* is a mathematical structure that represents relationships among entities in the real world.
- **incident edges** An edge in a graph that joins two vertices is said to be *incident* to both vertices.
- **parallel edge** A *loop* is an edge that links a vertex to itself. If two vertices are connected by two or more edges, these edges are called *parallel edges*.
- **Seven Bridges of Königsberg** The first known problem solved using graph theory.
- **simple graph** A simple graph is one that has no loops and parallel edges.
- **spanning tree** Assume that the graph is connected and undirected. A *spanning*

tree of a graph is a subgraph that is a tree and connects all vertices in the graph.

- **tree** A connected graph is a *tree* if it does not have cycles.
- **weighted graph** edges or vertices are assigned with weights.
- **undirected graph** no directed edges.
- **unweighted graph** edges are vertices are not assigned with weights.

Chapter 29

- **Dijkstra's algorithm** A well-known algorithm for finding the shortest path from a single source to all other vertices in a weighted graph.
- **edge-weighted graph** edges are assigned with weights.
- **minimum spanning tree** A spanning tree with the minimum total weights.
- **Prim's algorithm** A well-known algorithm for finding a spanning tree in a connected weighted graph.
- **complete graph** A complete graph is the one in which every two pairs of vertices are connected.
- **shortest path** a path between tow vertices with the shortest total weight.
- **single-source shortest path** a shortest path from a source to all other vertices.
- **vertex-weighted graph** Weights assigned to vertices.

Chapter 32

- **condition** A lock may create any number of Condition objects for thread communications.
- **deadlock** A situation in which two or more threads acquire locks on multiple objects and each has the lock on one object and is waiting for the lock on the other object.
- **event dispatcher thread** A designated thread for processing events in Java.
- **fail-fast** Refers to iterators. If you are using an iterator to traverse a collection while the underlying collection is being modified by another thread, then the iterator will immediately fail by throwing java.util.ConcurrentModificationException, which is a subclass of RuntimeException.
- **fairness policy** A parameter for a Lock. If fairness policy is true, it guarantees the longest-wait thread to obtain the lock first. False fairness policies grant a lock to a waiting thread without any access order. Programs using fair locks accessed by many threads may have poor overall performance than those using the default setting, but have smaller variances in times to obtain locks and guarantee lack of starvation.
- **lock** To access mutual exclusive resource, you must first obtain an appropriate lock.
- **race condition** A situation that causes data corruption due to unsynchronized access of data by multiple threads.
- **synchronization wrapper** The Collections class provides six static methods for wrapping a collection into a synchronized version.
- **synchronized** A keyword to specify a synchronized method or a synchronized block. A synchronized instance method acquires a lock on this object and a synchronized static method acquires a lock on the class. A synchronized block acquires a lock on a specified object, not just this object before executing the statements in the block.
- **thread** A flow of execution of a task, with a beginning and an end, in a program. A thread must be an instance of java.lang.Thread.
- **thread-safe** A class is said to be thread-safe if an object of the class does not cause a race condition in the presence of multiple threads.

Chapter 34

- **database system** consists of a database, the software that stores and manages data in the database, and the application programs that present data and enable the user to interact with the database system.
- **domain constraint** specifies the permissible values for an attribute. Domains can be specified using standard data types, such as integers, floating-point numbers, fixed-length strings, and variant-length strings. The standard data type specifies a broad range of values.
- **foreign key constraint** defines the relationships among relations. A foreign key is an attribute or a set of attributes in one relation that refers to the primary key in another relation.
- **integrity constraint** imposes a condition that all legal values of the tables must satisfy. In general, there are three types of constraints: *domain constraints*, *primary key constraints*, and *foreign key constraints*. DBMS enforces integrity constraints and rejects any operation that would violate them.
- **primary key constraint** specifies that the values of the primary key are unique in a relation.
- **relational database** based on the relational data model. A relational database stores data in tables (also known as relations). A relational data model has three key components: structure, integrity, and languages. *Structure* defines the representation of the data. *Integrity* imposes constraints on the data. *Language* provides the means for accessing and manipulating data.
- **Structured Query Language (SQL)** the language for defining tables and integrity constraints and for accessing and manipulating data.

Chapter 35

- **BLOB type** a new SQL type defined in SQL3 for representing a binary large object (e.g., an image file, java objects).
- **CLOB type** a new SQL type defined in SQL3 for representing a character large object (e.g., a large text file).
- **batch mode** executing SQL statements in batch mode from JDBC.
- **scrollable result set** You can scroll a result set in JDBC 2 and move the cursor in any direction or directly anywhere.
- **updateable result set** You can update database through a result set in JDBC 2.

Chapter 37

- **Common Gateway Interface** (CGI) A protocol for server-side programs to generate dynamic Web content.
- **CGI programs** The programs that interact with a Web server through the common gateway interface. The Web server receives a request from a Web browser and passes it to the CGI program. The CGI program processes the request and generates a response at runtime. CGI programs can be written in any language, but the Perl language is the most popular choice.
- **Cookie** Small text files that store sets of name-value pairs on the disk in the client's computer. Cookies can be used for session tracking.
- **GET and POST methods** The methods for sending requests to the Web server. The POST method always triggers the execution of the corresponding CGI or servlet program. The GET method may not cause the CGI or servlet program to be executed if the previous same request is cached in the Web browser.
- **HTML form** An HTML construct that enables you to submit data to the Web server in a convenient form. When issuing a request from an HTML form, either a GET method or a POST method can be used. The form explicitly specifies which of the two is used. If the GET method is used, the data in the form are appended to the request string as if it were submitted using a URL. If the POST method is used, the data in the form are packaged as part of the request file. The server program obtains the data by reading the file.
- **life-cycle methods** Every servlet implements the Servlet interface. The init, service, and destroy methods are known as life-cycle methods in the Servlet interface.
- **URL query string** Part of URL that specifies the location of the servlet program, parameters, and their values (e.g., ServletClass?pname1=pvalue1&pname2=pvalue2). The **?** symbol separates the program from the parameters. The parameter name and value are associated using the **=** symbol. Parameter pairs are separated using the **&** symbol. The **+** symbol denotes a space character.
- **servlet** A Java program that runs on a Web server to produce dynamic Web pages.
- **servlet container** (**servlet engine**) A software that runs servlets.
- **Tomcat** A servlet engine developed by Apache that serves as a standard reference implementation for Java servlets and Java Server Pages. It can be used to test servlets and ServerPages.