**abstraction**: the process of ignoring irrelevant details and emphasizing essential ones. To abstract is to disregard certain differentiating details.

**algorithm**: a set of instructions describing a pattern of behavior guaranteed to achieve a specific goal.

**component**: a well-defined part of a system.

**composition**: the process of building a system using simpler components.

**functionality**: the actions a program is expected to perform, or the purpose it is expected to satisfy.

**object**: a software abstraction representing some data component to be manipulated by the system.

**object-oriented**: a method of software development in which the system is organized around data objects.

**software system**: a collection of data descriptions and algorithms, expressed in a programming language, designed to solve a particular problem or set of related problems.

**command**: a constituent (feature) of an object used to instruct the object to perform some action which typically results in a change of the object’s state.

**external interface**: the subsystem responsible for communicating with the external world. See user interface.

**feature**: a query or command to which an object responds.

**immutable object**: an object that does not change state after it is created.

**instance**: an object. An object that is a member of a class is called an “instance of the class.”

**instance variable**: a variable that is a permanent part of an object; memory space for the variable is allocated when the object is created.

**model**: the subsystem responsible for representing and actually solving the problem at hand. Also known as the problem domain subsystem.

**mutable object**: an object that can change state after it is created.

**primitive type**: a type provided as an integral part of the language. Java’s built-in types include byte, short, int, long, float, double, char, and boolean.

**query**: a constituent (feature) of an object used to obtain some data value from the object. A query might simply provide the value associated with some property, or might require the object to perform a computation to produce the value.

**reference value**: a value which denotes or refers to an object.

**state**: the set properties of an object and their current values.

**system functionality**: the set of tasks a system is required to be able to perform.

**type**: a set of similar values along with the operations that can be performed with them.

**user interface**: the subsystem responsible for interacting with the human user of a system. See external interface.

**instance variable**: a variable that contains data stored as part of an object’s state. An instance variable is allocated when the object is created.

**interface**: the collection of features of an object as seen by its clients. Also called the specification of the object.

**package**: a set of related class definitions. A package is the fundamental structural unit of a Java application.server: in reference to a given object, a server is another object, which provides a service to the given object. The given object is called the client. The client accesses the features of the server, that is, it invokes the server’s methods.

**specification**: the collection of features of an object as seen by its clients. Also called the interface of the object.

**final feature**: a component or parameter that cannot be changed after it is given a value.
functional test: a test to determine if a system meets customer specifications.
unit test: a test to determine if a class is correctly implemented.
class invariant: an invariant regarding properties of class instances, that is, a condition that will always be true for all instances of a class.
guard: a boolean expression associated with a statement, that prevents or allows the statement to be executed. A guarded statement will be executed only if its guard evaluates to true. (See if-then statement.)
invariant: a condition that always holds.
postcondition: a condition the implementor of a method guarantees will hold when the method completes execution.
precondition: a condition the client of a method must make sure holds when the method is invoked.
assert statement: a statement that predicates the truth of a boolean condition. If the condition is true, the statement has no effect. If the condition is false, the statement causes an error exception which will terminate the program.
contract: part of a method specification that explicitly states the responsibility of the client and the responsibility of the server.
programming by contract: a programming style in which the invocation of a method is viewed as a contract between client and server, with each having explicitly stated responsibilities.
black box test: a test designed by considering only the expected behavior of a system, without regard for the internal implementation structure of the system.
equivalency group: a set of cases that are expected to behave identically for a test. If one member of an equivalency group passes a given test, all members should pass the test.
functional testing: testing a system to ensure that it conforms to the customer’s specifications.
gray box test: a test in which specification of the component classes is used to develop test cases.
implementation driven testing: a testing strategy in which the structure of the implementation is used to design the tests. See white box testing.
test driven implementation: an implementation strategy in which tests for a feature are written before the feature is implemented.
test fixture: a set of objects on which a test is to be performed.
test plan: a document that details how a system or system component will be tested.
unit testing: testing a system component to ensure that it is correct.
white box test: a test designed by taking into account the implementation of the system of the component being tested.
composition: the process of defining a new class by putting together existing classes. An instance of the new class, the composite, references instances of the existing classes, its components.
event-driven: characterizes a system designed to respond to external events, such as mouse clicks, button presses, menu choices, etc.
iteration: a process in which an operation is performed several times. More precisely, a problem solving technique in which the solution is obtained in a series of similar “steps,” each step building on the partial solution provided by the previous step.
architectural classes: classes that form the underlying structure of the solution.
functional specification: a description of the functionality required of a system. (See system specification.)implementation classes: classes that support the algorithmic implementation of the system.
iterative process: a procedure for solving a problem by repeated steps.
model classes: classes that directly model aspects of the external system.

problem analysis: a thorough examination of the problem to be solved, resulting in a system specification.

software life cycle: phases a software system undergoes from its inception to its retirement.

system design: defining a collection of classes and their interactions that will satisfy the system specifications.

system implementation: constructing the (software) modules that will make up the system, using programming languages and other software development tools.

system maintenance: modifying a system to meet changing requirements and correcting problems detected after the system is put into production.

system specification: a description of the functionality required of a system. (See functional specification.)

abstract method: a method specification in an interface. The method is specified but no implementation is provided.

cast: an operation by which an expression of one type is treated as an expression of a different type. A reference-valued expression can be cast to a supertype or a subtype.

generalization: the relationship of an interface to a more concrete class or interface.

instanceof: an operator that determines at run-time whether or not a reference value is of a specified type.

interface: a language structure that specifies functionality without any hint at implementation. An interface contains method specifications but no method implementations. It defines the functionality required by a client of a server.

is-a: the relationship of a class or interface to a more abstract interface.multiple inheritance: the ability of a class to implement than one interface, or of an interface to extend more than one interface.

observer pattern: a structural pattern by which a server informs a client that it (the server) has reached some particular state.

strategy pattern: a structural pattern by which an object is provided with a changeable component to which it delegates certain functionality.

subtype: a type whose constituent values are also members of another (super) type. If A is a subtype of B, then an expression of type A can be written in any context requiring a value of type B.

supertype: a type that contains the values of another (sub) type. (See subtype.)

dynamic binding: see polymorphism.

extends: the relationship of a class to a more abstract class, or of an interface to a more abstract interface. Also called is-a.

generalization: the relationship of a class or interface to a more concrete class or interface.

hierarchy: the structure of relationships that results from reiterated abstraction or extension.

inheritance: the mechanism by which a class or interface automatically gets all of the non-private features of its superclass or superinterface.

inner class: a nested class that is not a static member class.

is-a: the relationship of a class or interface to a more abstract class or interface. Also called extends.

local class: a named class defined in the body of a method or constructor.

member class: a nested class that is defined directly in the containing class. It is a feature of the containing class.

nested class: a class whose definition is containing in the body of another class.
overloading: providing a class with several distinct methods having the same name.
overriding: redefining the implementation of a method that a class inherits from its superclass.
package private: attribute of a feature that makes it accessible to all class members of same package.
polymorphism: dynamic behavior by which the algorithm performed as the result of a call to an overridden method is determined by the run-time class of the object executing the method. Also referred to as dispatching or dynamic binding.
protected: attribute of a features that makes it accessible in subclasses.
scope: the section of program text in which applied occurrences of an identifier refers to the identifier introduced by a particular definition.
scoping rules: language rules that associate applied identifier occurrences with defining occurrences.
single inheritance: the restriction that requires each class to have exactly one parent class.
subclass: the relationship of a class to the class it extends. The more concrete class is a subclass of the more abstract class.
superclass: the relationship of a class to a class that extends it. The more abstract class is a superclass of the more concrete class.
top level class: a class that is not a nested class.
abstract class: a class that can contain abstract methods and can’t be instantiated. The purpose of an abstract class is to serve as a foundation on which to build subclasses.
base class: a class that is extended to define a subclass.
concrete class: a class that contains a complete implementation, and is instantiated to provide runtime objects that participate in the problem solution.
function object: an object whose only purpose is to carry some functionality. Typically, a function object has no state.