abstract class
A class whose primary purpose is to define an interface. An abstract class defers some or all of its implementation to subclasses. An abstract class cannot be instantiated.

abstract coupling
Given a class A that maintains a reference to an abstract class B, class A is said to be abstractly coupled to B. We call this abstract coupling because A refers to a type of object, not a concrete object.

abstract operation
An operation that declares a signature but doesn't implement it. In C++, an abstract operation corresponds to a pure virtual member function.

acquaintance relationship
A class that refers to another class has an acquaintance with that class.

aggregate object
An object that's composed of subobjects. The subobjects are called the aggregate's parts, and the aggregate is responsible for them.

aggregation relationship
The relationship of an aggregate object to its parts. A class defines this relationship for its instances (e.g., aggregate objects).

black-box reuse
A style of reuse based on object composition. Composed objects reveal no internal details to each other and are thus analogous to "black boxes."

class
A class defines an object's interface and implementation. It specifies the object's internal representation and defines the operations the object can perform.

class diagram
A diagram that depicts classes, their internal structure and operations, and the static relationships between them.

class operation
An operation targeted to a class and not to an individual object. In C++, class operations are called static member functions.

concrete class
A class having no abstract operations. It can be instantiated.
**constructor**
In C++, an operation that is automatically invoked to initialize new instances.

**coupling**
The degree to which software components depend on each other.

**delegation**
An implementation mechanism in which an object forwards or delegates a request to another object. The delegate carries out the request on behalf of the original object.

**design pattern**
A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context.

**destructor**
In C++, an operation that is automatically invoked to finalize an object that is about to be deleted.

**dynamic binding**
The run-time association of a request to an object and one of its operations. In C++, only virtual functions are dynamically bound.

**encapsulation**
The result of hiding a representation and implementation in an object. The representation is not visible and cannot be accessed directly from outside the object. Operations are the only way to access and modify an object's representation.

**framework**
A set of cooperating classes that makes up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes.

**friend class**
In C++, a class that has the same access rights to the operations and data of a class as that class itself.

**inheritance**
A relationship that defines one entity in terms of another. **Class inheritance** defines a new class in terms of one or more parent classes. The new class inherits its interface and implementation from its parents. The new class is called a **subclass** or (in C++) a **derived class**. Class inheritance combines **interface inheritance** and **implementation inheritance**. Interface inheritance defines a new interface in terms of one or more existing interfaces. Implementation inheritance defines a new implementation in terms of one or more existing implementations.
instance variable
A piece of data that defines part of an object's representation. C++ uses the term data member.

interaction diagram
A diagram that shows the flow of requests between objects.

interface
The set of all signatures defined by an object's operations. The interface describes the set of requests to which an object can respond.

metaclass
Classes are objects in Smalltalk. A metaclass is the class of a class object.

mixin class
A class designed to be combined with other classes through inheritance. Mixin classes are usually abstract.

object
A run-time entity that packages both data and the procedures that operate on that data.

object composition
Assembling or composing objects to get more complex behavior.

object diagram
A diagram that depicts a particular object structure at run-time.

object reference
A value that identifies another object.

operation
An object's data can be manipulated only by its operations. An object performs an operation when it receives a request. In C++, operations are called member functions. Smalltalk uses the term method.

overriding
Redefining an operation (inherited from a parent class) in a subclass.

parameterized type
A type that leaves some constituent types unspecified. The unspecified types are supplied as parameters at the point of use. In C++, parameterized types are called templates.

parent class
The class from which another class inherits. Synonyms are superclass (Smalltalk), base class (C++), and ancestor class.

polymorphism
The ability to substitute objects of matching interface for one another at run-time.
**private inheritance**
In C++, a class inherited solely for its implementation.

**protocol**
Extends the concept of an interface to include the allowable sequences of requests.

**receiver**
The target object of a request.

**request**
An object performs an operation when it receives a corresponding request from another object. A common synonym for request is **message**.

**signature**
An operation’s signature defines its name, parameters, and return value.

**subclass**
A class that inherits from another class. In C++, a subclass is called a **derived class**.

**subsystem**
An independent group of classes that collaborate to fulfill a set of responsibilities.

**subtype**
A type is a subtype of another if its interface contains the interface of the other type.

**supertype**
The parent type from which a type inherits.

**toolkit**
A collection of classes that provides useful functionality but does not define the design of an application.

**type**
The name of a particular interface.

**white-box reuse**
A style of reuse based on class inheritance. A subclass reuses the interface and implementation of its parent class, but it may have access to otherwise private aspects of its parent.