**Key Java Terms and Concepts**

**Conditional Statements and Repetition Statements: will not be as prominent as they were on Exam 2, but you should be familiar with the basic concepts, and know how the if(-else), while, and for loops function.**

**The four object-oriented principles are abstraction, encapsulation, inheritance, and polymorphism. Inheritance and polymorphism will be most prominent on the exam.**

**static**: keyword establishing that a **variable** or **method** is associated with a class, rather than an individual object. Do not confuse static with final.

**final**: keyword establishing limitations on the mutability of a variable, method, or class. A **final variable** cannot be modified after it is initialized. A **final method** cannot be overridden in children classes (if they exist). Other classes *cannot* inherit from a **final class**.

**public**: keyword indicating a variable, method, or class can be used without restriction by outside class code (assuming proper import statements, etc. are used if necessary). Public variables violate **encapsulation**, and are often avoided.

**private**: keyword indicating a variable, method, or class can only be directly used within the scope of its class definition. **Accessor** methods are public methods that return the value of a private variable, while **mutator** methods are public methods that modify a private variable.

**protected**: a keyword indicating a variable, method or class can be used directly within the scope of its class definition, as well as within the definition of any inheriting class. By convention, protected entities are also available within classes in the same package.

**Dependency**: a class relationship wherein one class **uses** another for a specific operation. Ex: a *Doctor* object may use a *Scalpel* object within method *surgery*.

**Aggregation**: a class relationship wherein one class fundamentally **has** another as a principal portion of its identity. Ex: a *Book* object may have/possess a number of *Page* objects.

**Inheritance**: a class relationship wherein a **child class** inherits (automatically obtains) the instance variables and methods of a **parent class**, using the keyword **extends**. Ex: a *Supervisor* class may inherit from an *Employee* class. Do not confuse inheritance with interfaces.

**Abstract Class**: A class that cannot be directly instantiated, and is instead created to provide a blueprint for inheriting classes. May have **abstract methods** (header declared but no body), as well as defined methods and variables.

**Interface**: A collection of abstract methods that establish what operations any class that **implements** it must define.

*Remember: a class may implement multiple interfaces, but each may extend (inherit from) only one class. Any class without an “extends” clause automatically inherits from the* ***Object*** *class.*

**Abstract Class:** a class that cannot be instantiated, and instead serves as a “template” for other classes via inheritance. Can have both abstract and non-abstract methods, as well as variables.

**super:** A keyword used to either invoke a parent class’s constructor, or one its other methods. Needed in part because *constructors are not inherited*.

**Method Overloading**: giving a single method name multiple definitions in a class, which must vary according to input parameters (type and/or quantity). Do not confuse method overloading with method overriding.

**Method Overriding**: giving an inheriting class a distinct method definition for the exact same input arguments, presumably to account for differing behavior between child and parent class.

**Array**: A data structure representing a collection of primitive or object data, which has fixed size when created and is relatively efficient for storage and access speed. Do not confuse the standard array with the ArrayList class.

An **abstract data type** (ADT) is an implementation-independent description of a structure’s operations, while a data **structure** refers to an implemented version.

**Queues** are a first-in, first-out (FIFO) ADT, while **stacks** are a last-in, first-out (LIFO) ADT.

**ArrayList**: A java.util data structure class that is used to represent a collection of object data, which has dynamic size and slightly greater overhead (slightly less efficient) than the array**. Wrapper classes** (Integer, Double, etc.) are needed if primitive-style data are required.

**LinkedList**: A Java Collection data structure class that is a *dynamically-linked* alternative to the ArrayList. Memory is dynamically allocated in the form of *nodes*, which include data and references to subsequent items (if any).

**Polymorphic reference**: a variable that can reference different types of variables at different points in time. Java permits both inheritance-based and interface-based polymorphism.

**Inheritance-based polymorphism**: A variable of a class type can reference an object of that type or any class in an *inheritance chain* from that class.

**Interface-based polymorphism**: A variable of an interface type can reference any object that implements it.

**Recursive method:** a method that invokes itself either directly or indirectly (make sure you understand the difference between recursion and iteration)

**Sorting**: The process of arranging a collection of items in a hierarchical order (ex: increasing #)

**Searching:** The process of searching for a particular item within a collection of items. **Binary search** requires a sorted collection but is more efficient for large collections than **linear search**.

**Generics** (ex: **<T>** in **public class myClass <T>** {…}) allow us to efficiently program classes and methods that accept **multiple data-types** in performing their operations.