What is an **object?**

**object** has its own set of *data*, together with a set of *methods* that act upon the data. An **object** stores its data in **instance variables**

**Object** example: String, System.out , Scanner. **Method** example: length, substring.

What is a **Class**?

A **class** describes a set of *object*s with the same behavior.

Example of a class: String, the String class specifies how a string *store*s its characters, which *method*s can be used with strings, and how the methods are *implemente*d.

Every class has a **public interface**: a collection of methods through which the objects (in which contains data) of the class can be manipulated.

The process of providing a public interface, while hiding the implementation details, is called **encapsulation**.

Encapsulation enables changes in the implementation without affecting users of a class.

Java use the ***new*** operator to construct objects

Counter tally = new Counter();

Class name object name

To invoke the count method: tally.count();

this is the method

the *tally* object remembers the effect of prior method calls.

To implement a class:

First specify how each object stores its data, eg: int, double, String, etc.

An object stores its data in ***instance variables***

Example: private int value;

(modifier)

Each object of a class has its own set of instance variables

Then implement the methods in the class.

public void count(){ value = value + 1;}

method name: count, no parameter. Can have parameter.

The methods that you invoke on an object are called **instance methods.** An instance method can access the instance variables of the object on which it acts. In the above case is “value”.

A private instance variable can only be accessed by the methods of its own class.

Private instance variables are an essential part of encapsulation

**public interface** of a class consists of all methods that a user of the class may want to apply to its objects

methods of a class are instance methods. They are not declared as *static*.

An **accessor** (getter) method does not change the object on which it operates.

For each accessor method, an object must either store or compute the result.

A **mutator** (setter) method changes the object on which it operates.

Be sure that your data representation supports method calls in any order.

The object on which a method is applied is the implicit parameter.

The object on which a method is applied is the **implicit parameter**

The object on which a method is invoked is called the implicit parameter of the method. In Java, you do not actually write the implicit parameter in the method declaration

**Explicit parameters** of a method are listed in the method declaration

parameters that are explicitly mentioned in the method declaration, are called explicit parameters.

Every method has exactly one implicit parameter and zero or more explicit parameters.

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.

Eg: Scanner in = new Scanner(System.in);

Constructors never return values, but you do not use the *void* reserved word when declaring them.

public CashRegister() // A constructor

A class can have multiple constructors.

The compiler picks the constructor that matches the construction arguments

If you do not initialize an instance variable in a constructor, it is automatically set to a default value: 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, so that every class has at least one constructor.

A unit test verifies that a class works correctly in isolation, outside a complete program.

you can test a class simply by constructing an object, calling methods, and verifying that you get the expected return values.

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.

An **object reference** specifies the memory 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.

CashRegister reg1 = new CashRegister();

CashRegister reg2 = reg1;

reg2.addItem(2.95);

The *null* reference refers to no object.

String middleInitial = null;

Note that the null reference is not the same as the empty string “”.

In a method, the ***this*** reference refers to the implicit parameter.

static variable belongs to the class, not to any object of the class.

private static int lastAssignedNumber = l000;

static constants may be either private or public

public static final double OVERDRAFT\_FEE = 29.95;

A static method is not invoked on an object. Not with the *new* operator.

the Math class provides a static method that is invoked as Math.sqrt(x).

public static double percentOf(double percentage, double amount)

{ return (percentage / 100) \* amount; }

double tax = Financial.percentOf(taxRate, total);