



# How To Design program



# What is Program?

- A computer program is a set of ordered instructions for a computer to perform a specific task or to exhibit desired behaviors.
- Without programs, computers are useless.
- A software application or software program is the most commonly found software on the computer.
  - Microsoft Word is a word processor program that allows users to create and write documents.



# What is Programming?

- **Computer programming** is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs.
- **Programming language:**
  - The source code of program is written as a series of human understandable computer instructions in a that can be read by a **compiler**, and translated into machine code so that a computer can understand and run it.
  - There are many programming languages such as C++, C#, Java, Python, Smalltalk, etc.



# Design Program

- The process of programming often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.
- Main goal of the **design process** is to lead from **problem statements** to **well-organized solutions**
- Design guidelines are formulated as a number of ***program design recipes***.
  - A design recipe guides a beginning programmer through the entire problem-solving process



# How to Design Program

To design a program properly, a programmer must:

1. **Analyze** a problem statement, typically stated as a word problem;
2. **Express** its essence, abstractly and with examples;
3. **Formulate** statements and comments in a precise language;
4. **Evaluate** and revise these activities in light of checks and tests; and
5. Pay **attention** to **details**.



# Why Java?

- Object-oriented programming languages
- Open source
- A cross platform language
  - Portability: *"Write Once, Run Anywhere"*
- Spread Rapidly through WWW and Internet
- **JVM** (Java Virtual Machine)
- **JRE** (Java Runtime Environment)
- **JDK** (Java Developer Kit)



# IDE

- **IDE**: **I**ntegrated **D**evelopment **E**nvironment
  - **Netbean**: supported by Sun
  - **Eclipse**: open source, supported by IBM



# The Varieties of Data





# Topic

- Primitive Forms of Data
- Compound Data: Class
  - Design Class
- Class References, Object Containment
- Design method



# Primitive Forms of Data

- Java provides a number of built-in **atomic forms of data** with which we represent primitive forms of information.



# Integer type

Name	Size	Range
byte	1 byte	-128 ... 127
short	2 bytes	-32,768 ... 32,767
int	4 bytes	-2,147,483,648 to 2,147,483,648
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808



# Decimal type

Name	Size	Range
float	4 bytes	$\pm 1.4 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
double	8 bytes	$\pm 4.9 \times 10^{-324}$ to $\pm 1.8 \times 10^{308}$



# Character type

Java represent a Unicode character (2 bytes)

Name	Size	Range
<code>char</code>	2 bytes	<code>\u0000 ... \uFFFF</code>

- **Q:** Distinguish char type and String type ?
- **A:**
  - `char c = 't';`
  - `String s = "tin hoc";`



# Boolean type

Name	Size	Range
<code>boolean</code>	1 byte	<code>false, true</code>



# String

- ***Strings*** to represent symbolic.  
Symbolic information means the names of people, street addresses, pieces of conversations, ...
- a *String* is a sequence of keyboard characters enclosed in quotation marks
  - "bob"
  - "#\$%^&"
  - "Hello World"
  - "How are U?"
  - "It is 2 good to B true."



# Compound Data: Class

- For many programming problems, we need more than atomic forms of data to represent the relevant information.





# Coffee Example

Consider the following problem:

- Develop a program that keeps track of coffee sales at a specialty coffee seller. The sales receipt must include the kind of coffee, its price (per pound), and its weight (in pounds).



# Coffee sale information

- The program may have to deal with hundreds and thousands of sales.
- We need to **keep all pieces of information about a coffee sale together in one place.**
- The information for a coffee sale consists of three (relevant) pieces: the **kind of coffee**, its **price**, and its **weight**. For example, the seller may have sold:
  - 1) 100 pounds of Hawaiian Kona at \$15.95/pound
  - 2) 1,000 pounds of Ethiopian coffee at \$8.00/pound
  - 3) 1,700 pounds of Colombian Supreme at \$9.50/pound

# Define Java Class, Constructor

- We would have used a class of structures to represent such coffee sales: **class**
  - a **coffee** class has three fields: **kind**, **price**, and **weight**.
  - the **constructor** is to get the values of the three fields

```
class Coffee {  
    String kind;  
    double price;  
    double weight;
```

← **class definition**

← **Field declarations.** Each declaration specifies the type of values that the field name represents.

```
    Coffee(String kind, double price, double weight) {  
        this.kind = kind;  
        this.price = price;  
        this.weight = weight;  
    }  
}
```

← **CONSTRUCTOR  
of the class**

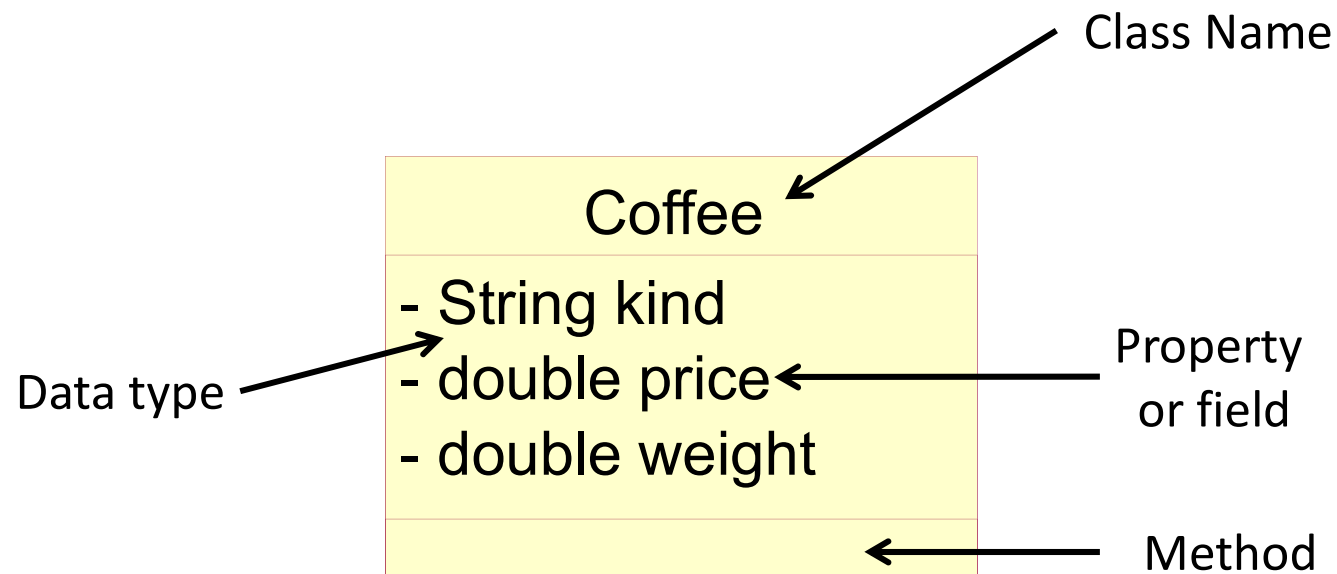


# About Constructor

- Constructor name is same class name.
- The parameters of a constructor enclose in () quotes, separated by commas (,).
- Its body is a semicolon-separated (;) series of statement `this.fieldname = parametername;`

```
Coffee(String kind, double price, double weight) {  
    this.kind = kind;  
    this.price = price;  
    this.weight = weight;  
}
```

# Class Diagram: abstractly express





# Translate sample

- It is best to translate some sample pieces of information into the chosen representation.
- This tests whether the defined class is adequate for representing some typical problem information.
- Apply the constructor to create an **object** (**instance**) of the **Coffee** class:
  - **new** *Coffee*("Hawaiian Kona", 15.95, 100)
  - **new** *Coffee*("Ethiopian", 8.00, 1000)
  - **new** *Coffee*("Colombia Supreme", 9.50, 1700)

# Test Class

Import test unit library

```
import junit.framework.*;
```

Define class CoffeeTest is test case.

```
public class CoffeeTest extends TestCase {
```

Test method that name is testXXX

```
    public void testConstructor() {  
        new Coffee("Hawaiian Kona", 15.95, 100);  
        new Coffee("Ethiopian", 8.0, 1000);  
        new Coffee("Colombian Supreme ", 9.5, 1700);  
    }  
}
```

This tests contain statement to create an object of the **Coffee** class, you apply the constructor to as many values as there are parameters:  
`new Coffee("Hawaiian Kona", 15.95, 100)`



# Date example

- Develop a program that helps you keep track of daily. One date is described with three pieces of information: a **day**, a **month**, and a **year**
- Class diagram







# Define Class, Constructor and Test

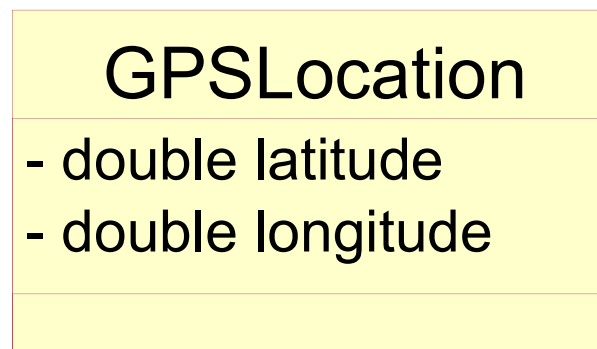
```
class Date {  
    int day;  
    int month;  
    int year;  
    Date(int day, int month, int year) {  
        this.day = day;  
        this.month = month;  
        this.year = year;  
    }  
}
```

```
import junit.framework.*;  
public class DateTest extends TestCase {  
    public void testConstrutor() {  
        new Date(5, 6, 2003); // denotes June 5, 2003  
        new Date(6, 6, 2003); // denotes June 6, 2003  
        new Date(23, 6, 2000); // denotes June 23, 2000  
    }  
}
```



# GPS Location example

- Develop a GPS navigation program for cars. The physical GPS unit feeds the program with the current location. Each such location consists of two pieces of information: the **latitude** and the **longitude**.
- Class diagram





# Define Class, Constructor and test

```
class GPSLocation {  
    double latitude;    /* degrees */  
    double longitude;   /* degrees */  
    GPSLocation(double latitude, double longitude) {  
        this.latitude = latitude;  
        this.longitude = longitude;  
    }  
}
```

```
import junit.framework.*;  
public class GPSLocationTest extends TestCase {  
    public void testConstructor() {  
        new GPSLocation(33.5, 86.8);  
        new GPSLocation(40.2, 72.4);  
        new GPSLocation(49.0, 110.3);  
    }  
}
```



# Three steps in designing Classes

1. Read the problem statement.

Look for statements that mention or list the attributes of the objects in your problem space.

Write down your findings as a class diagram because they provide a quick overview of classes.

2. Translate the class diagram into a class definition, adding a purpose statement to each class.

3. Obtain examples of information and represent them with instances of the class.

Conversely, make up instances of the class and interpret them as information.



# Class References, Object Containment

- Sometimes information not only consist of several pieces of information, but one of its constituents consists of several pieces, too.



# Runner's training log

Take a look at this problem:

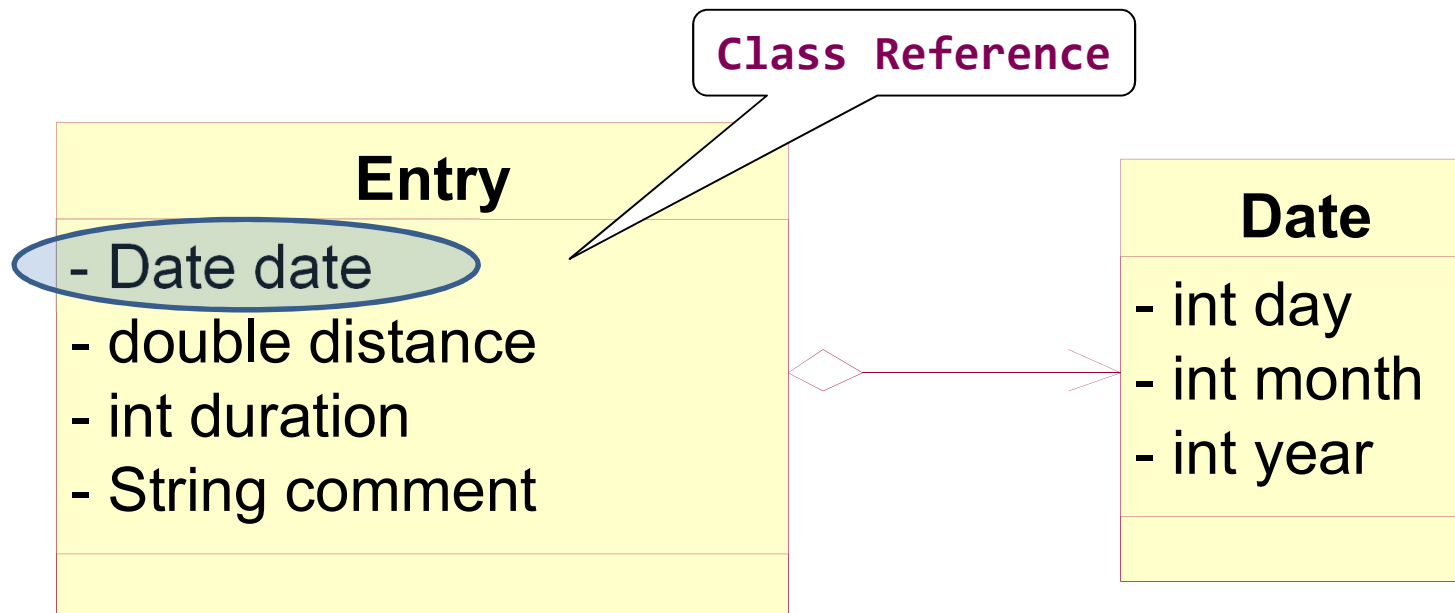
- Develop a program that manages a runner's training log. Every day the runner enters one entry concerning the day's run. Each entry includes the day's **date**, the **distance** of the day's run, the **duration** of the run, and a **comment** describing the runner's post-run feeling.



# Log Entry examples

- A log entry consists of four pieces of information: a date, a distance, a duration, and a comment.
  - To represent the last three, we can use primitive types; double (in miles), int (in minutes), and String
  - Representation for dates consists of three pieces: day, month, year
- Examples:
  - on June 5, 2003: 5.3 miles in 27 minutes, feeling good;
  - on June 6, 2003: 2.8 miles in 24 minutes, feeling tired
  - on June 23, 2003: 26.2 miles in 150 minutes, feeling exhausted;

# Class Diagram





# Define class and constructor

```
class Entry {  
    Date date; class reference  
    double distance;  
    int duration;  
    String comment;  
    Entry(Date date, double distance, int duration,  
           String comment) {  
        this.date = date;  
        this.distance = distance;  
        this.duration = duration;  
        this.comment = comment;  
    }  
}
```

```
class Date {  
    int day;  
    int month;  
    int year;  
    Date(int day, int month,  
         int year) {  
        this.day = day;  
        this.month = month;  
        this.year = year;  
    }  
}
```

# Test constructor

object containment

```
import junit.framework.*;
public class EntryTest extends TestCase {

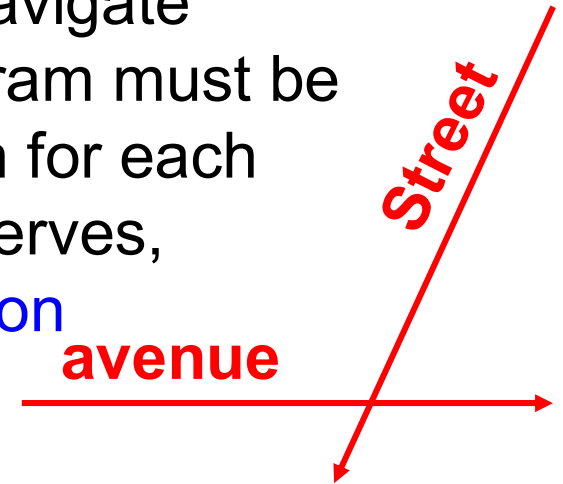
    public void testConstructor(){
        new Entry(new Date(5, 6, 2004), 5.3, 27, "good");

        new Entry(new Date(6, 6, 2004), 2.8, 24, "tired");

        Date date1 = new Date(23, 6, 2004);
        new Entry(date1, 26.2, 159, "exhausted");
    }
}
```

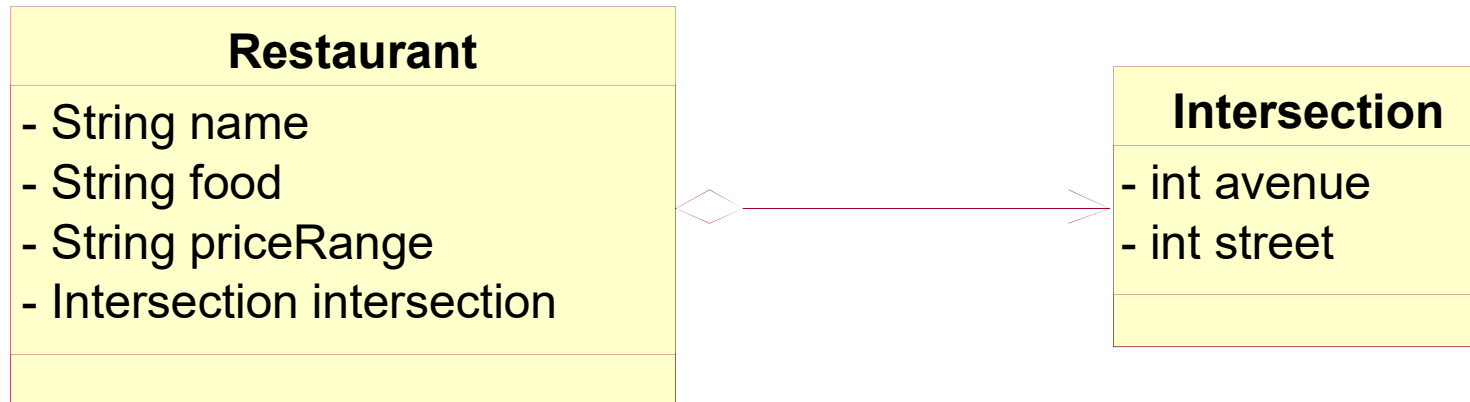
# Restaurant example

- Develop a program that helps a visitor navigate Manhattan's restaurant scene. The program must be able to provide four pieces of information for each restaurant: its **name**, the kind of **food** it serves, its **price** range, and the closest **intersection** (street and avenue).



- Examples:
  - La Crepe, a French restaurant, on 7th Ave and 65th Street, moderate;
  - Bremen Haus, a German restaurant on 2nd Ave and 86th Street, moderate;
  - Moon Palace, a Chinese restaurant on 10th Ave and 113th Street, inexpensive;

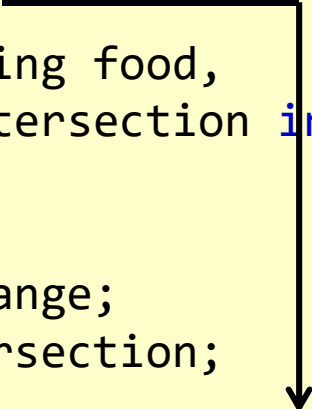
# Class Diagram



# Define class and constructor

```
class Restaurant {  
    String name;  
    String food;  
    String priceRange;  
    Intersection intersection;  
    Restaurant(String name, String food,  
                String priceRange, Intersection intersection) {  
        this.name = name;  
        this.food = food;  
        this.priceRange = priceRange;  
        this.intersection = intersection;  
    }  
}
```

reference

A black arrow originates from the 'intersection' parameter in the Restaurant constructor signature and points down to the 'Intersection' class definition box.

```
class Intersection {  
    int avenue;  
    int street;  
    Intersection(int avenue, int street) {  
        this.avenue = avenue;  
        this.street = street;  
    }  
}
```



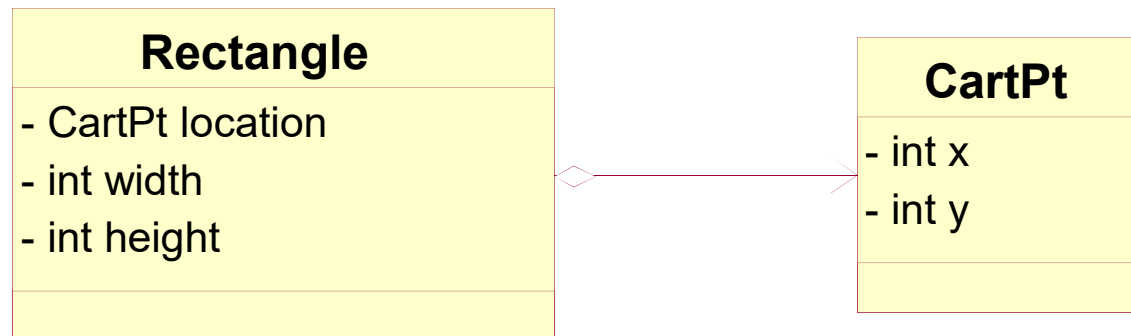
# Test constructor

```
import junit.framework.*;
public class RestaurantTest extends TestCase {

    public void testConstructor() {
        new Restaurant("La Crepe", "French", "moderate",
                       new Intersection(7, 65));
        new Restaurant("Bremen Haus", "German", "moderate",
                       new Intersection(2, 86));
        Intersection intersection1 = new Intersection(10, 113);
        new Restaurant("Moon Palace", "Chinese", "inexpensive",
                       intersection1);
    }
}
```

# Rectangle example

- The rectangles have **width**, **height** *and are located* on the Cartesian plane of a computer canvas, which has its origin in the northwest corner.





# Define class and constructor

```
class CartPt {  
    int x;  
    int y;  
    CartPt(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

```
class Rectangle {  
    CartPt location;  
    int width;  
    int height;  
    Rectangle(CartPt location, int width, int height) {  
        this.location = location;  
        this.width = width;  
        this.height = height;  
    }  
}
```





# Test constructor

```
import junit.framework.*;

public class RectangleTest extends TestCase {

    public void testContructor() {
        CartPt p = new CartPt(3, 4);
        CartPt q = new CartPt(5, 12);

        new Rectangle(p, 5, 17);
        new Rectangle(q, 10, 10);
        new Rectangle(new CartPt(4, 3), 5, 12);
    }
}
```