第3章

从GPSS到GPSS/Java

3.1 GPSS 建模基础

3.1.1 GPSS 处理程序

GPSS 处理程序亦称为 GPSS 处理器，实际就是 GPSS 解释程序。与一般算法语言的解释程序唯一的区别是 GPSS 解释程序实现了 GPSS 建模方法所基于的仿真算法。

3.1.2 GPSS 描述模型的方法

模块图：描述模型的逻辑结构，相当于程序框图或流程图，一共有 50 块左右，不同形状的模块代表不同的模拟功能。模块之间用箭线连接，箭头所指模块表示当前模块执行完后下一个要执行的模块。通过模块首尾相接的“箭线”定义了模块执行的顺序，也称为模块图路径。

模型程序：由按一定的逻辑排列并遵守一定语法规则的 GPSS 模块语句（一个模块语句对应一个模块图）和控制语句组成的文本文件，可由 GPSS 解释器执行。

例如，有如下问题：某理发馆只有一位理发师，顾客到达时间间隔平均为 18 分钟，偏差 6 分钟，均匀分布。理发师为一个顾客理发平均所需时间为 16 分钟，偏差 4 分钟，均匀分布。顾客到达后，若理发师闲，则立刻接受服务，否则按先来先接受服务的规则排队等待，模拟 8 小时。

(1) 用模块图描述模型如图 3-1 所示。

(2) GPSS(H)模型程序如下。

```
 * Demo3_1.gps
SIMULATE
GENERATE        18,6
QUEUE           JOEQ
SEIZE           JOE
DEPART          JOEQ
ADVANCE         16,4
```

```
RELEASE        JOE
TERMINATE
GENERATE       480
TERMINATE      1
START          1
END
```

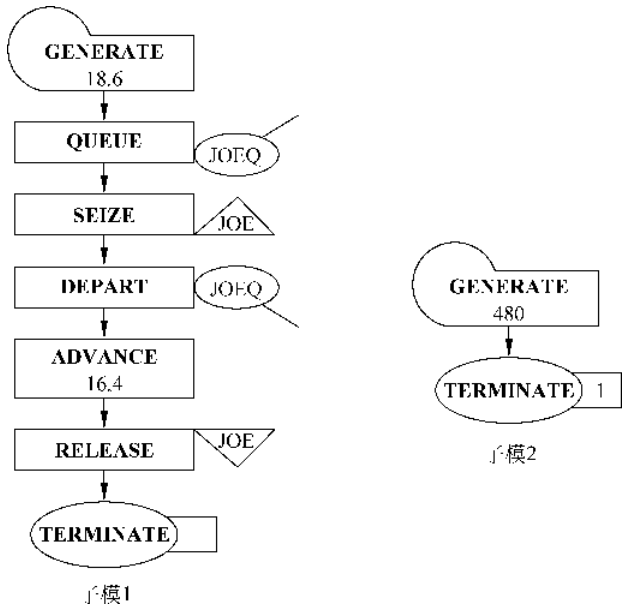

图 3-1　GPSS 模块图

3.1.3　动态实体、服务实体和辅助实体

实体是实际存在的事物，也称元素，是构成系统的基本事物。对于排队系统，实体可以分为两大类，一类为接受服务的实体，另一类为提供服务的实体。GPSS 模型中的实体与大多数实际系统中的实体具有直接的对应。

1. 动态实体

在 GPSS 模型中，动态实体定义为沿模型路径（模块图中的箭线）移动的运动体，代表实际系统中接受服务的实体。每个动态实体都有一组为实现仿真算法而设置的属性，如当前所在模块的位置，下一个要进入的模块位置等，并不对应实际事物的属性。同时还允许建模者定义一组属性，来描述每个动态实体的特性，这组属性通常与实际事物的属性对应与联系。

动态实体也称为流动实体或临时实体，流动表述了其运动性，因为一个动态实体总是试图沿模型路径从一个模块流入下一个模块；临时表述了其在某些模型中的暂驻性，在接受完所有服务之后，动态实体所代表的实体将离开系统。

例如，在模拟理发馆系统的 GPSS 模型中，每一个动态实体就代表该系统中的一个顾

客。顾客在此系统发生了如下的流动：顾客到达理发馆→顾客排入队列→顾客接受服务→顾客离开理发馆，如表 3-1 所示。

表 3-1 动态实体模型

真 实 系 统	动态实体代表的实际事物	真 实 系 统	动态实体代表的实际事物
超级市场	顾客	加工车间	零件
高速公路	汽车	就诊医院	病人
货运系统	卡车		

动态实体在模型中，具有以下行为特点。

(1) 根据需要按一定时刻或一定间隔时间进入模型。

(2) 在某一时刻可以离开模型。

(3) 众多动态实体可以同时存在于模型中。

(4) 在某一时刻，若有多个动态实体都要在模型中向前移动，那么要采取依次处理的机制。

动态实体总是试图从一个模块流入另一个模块，只有下列条件之一发生时停止运动。

(1) 流入的模块具有使动态实体停留某一事先确定的间隔时间的功能。

(2) 流入的模块具有拒绝动态实体流入的功能，此时动态实体停留在其紧前模块。

(3) 动态实体流入 TERMINATE 模块，便被排出模型之外。

2. 服务实体

在 GPSS 模型中所定义的服务实体映射为实际系统中能够提供服务的实体。同样每个服务实体也有两组属性。一组用于实现算法，另一组来描述服务实体当前的状态。

服务实体亦称为永久实体或固定实体。服务实体的称谓描述了其服务功能，从模型的动态角度来观察，在模型运行期间，这些提供服务的实体不会随着时间的变化而发生移动或消失，它们具有相对的固定性和永久性。

服务实体提供服务，那么就存在服务能力问题。服务实体的服务能力，也称为服务实体容量，即一个服务实体能最多同时提供多少个相同的服务。GPSS 只定义了两种类型的服务实体：一种称为设备实体(简称设备)，另一种称为存储实体(简称存储器)。设备实体在任一时刻只能提供一个服务，而存储实体可以提供多个服务。容量为 N 的存储实体，可以同时提供“N 个”服务。

例如，在模拟理发馆系统的 GPSS 模型中，JOE 代表了模型定义的一个服务实体(设备实体)，JOE 映射只有一位理发师的理发馆系统，它向顾客提供理发服务，同一时刻它只能为一位顾客提供理发服务，如表 3-2 所示。

问题是，设备实体与容量为 1 的存储实体又有何区别呢？从服务能力来看，两者没有区别，但是在功能方面却有区别。GPSS 定义的设备实体允许被抢占，而存储实体不允许。所谓抢占是指设备暂时中断当前的服务，而开始为更紧急的任务提供服务。抢占设备的有关概念与问题将在第 8 章讨论。

表 3-2　服务实体模型

真 实 系 统	服务实体代表的实际事物	真 实 系 统	服务实体代表的实际事物
超级市场	收费员	加工车间	加工机床
高速公路	收费站	就诊医院	医生与化验师
货运系统	仓库		

服务实体提供服务，而服务总要持续一段时间。每次服务时间可以是相同的确定值，也可以是服从某种分布的随机数值。

由以上定义和叙述可见：GPSS 模型系统中的动态实体和服务实体是对构成实际系统的基本事物的抽象与映射。动态实体代表接受服务的事物，而服务实体代表提供服务的事物；这两类实体的相互联系和作用代表了实际事物之间的联系和作用；而在模型中动态实体与服务实体的联系和作用正是通过将服务实体与动态实体相耦合的模块语句来实现的。这种模块，在面向对象技术中称为消息。从建模者的视角，当我们说一个动态实体沿路径流入某一模块，而对于 GPSS 处理程序来讲，就是按照预定的顺序调用与执行该模块对应的函数（或过程或方法）。在执行某个与服务实体相关的模块函数中，处理程序可以通过调用时传递的信息，来选择所要操作的服务实体和动态实体，并可根据它们或其他实体的当前状态属性来进一步选择不同的操作。处理器程序的判断、选择和执行的操作映射了实际系统中主客体事物的行为。因此可见 GPSS 建模的指导思想是建立在面向对象分析的基础之上的。而当今在系统工程和软件工程中被广泛使用的面向对象技术正是来源于系统仿真的研究。

3. 辅助实体

GPSS 定义了若干不同类型的辅助实体，大都不对应实际系统的具体事物。这类实体有排队实体（简称队列）、函数实体（简称函数）、保存值实体（简称保存值或保留值）、表实体（简称表或统计表）、逻辑开关实体（简称逻辑开关或开关）、变量实体（简称变量）和用户链实体（简称用户链）等。它们为建模提供逻辑、计算、统计和输出等方面的辅助功能。从定义这些辅助实体的数据结构方面来看，很类似于服务实体，因此在模型中将服务实体和辅助实体统称为资源实体，但读者应该明确它们之间的本质区别。

3.1.4　模拟时钟和仿真算法

GPSS 定义了两个变量（全局变量）用来记录模型运行时一系列事件发生时的相对时间（相对于模拟开始时），两个变量分别称为绝对时钟和相对时钟，初始值（模型开始运行时）都取 0。模型运行的过程中，随着新的事件的不断发生，绝对时钟（相对时钟以后介绍）不断更新，时钟值不断向前推移（增大）。对于离散系统，时钟值的变化肯定也是离散的，因为前后相邻的事件发生在具有间隔的时间段的两端。如果绝对时钟当前记录的是相邻前一事件发生时的时间，于是处理程序在完成该事件发生时所对应的各种操作（修正系统各状态属性，简称系统状态修正）后，将当前时钟值加上相邻紧后一事件的发生间隔（最短时间步长），来修正时钟（时钟修正），此时时钟即为相邻后一事件的发生时间。此时的事件称为当前事件，此时的时钟时间称为当前时间。同样处理程序在刚刚修正时钟后的当前时间，立即执行当

前事件所对应的各种操作(系统状态修正)。当然不同的事件会对应不同的操作。以上算法被称为最短时间步长事件模拟法(图 3-2 为该方法示意图)。

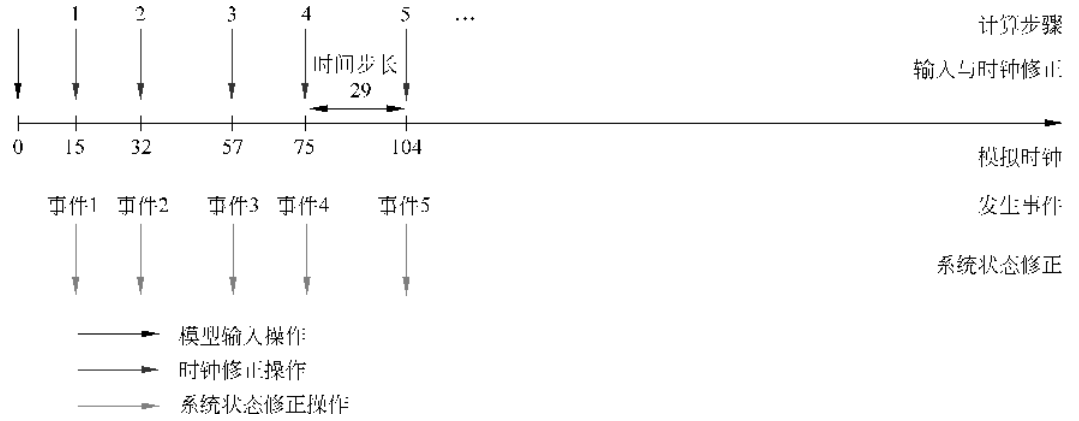

图 3-2 最短时间步长事件模拟法示意图

与面向事件建模方法有所不同的是：GPSS 的仿真算法中，在关键事件发生时，处理程序完成时钟修正后，是按照代表事件发生的模块的执行顺序(沿模型路径)来模块化地修正系统状态属性的，而这些代表普通事件或关键事件发生的模块的命名与相互关联的设计使模型非常贴近现实的管理系统。模型中的元素和流程与现实系统的事物和业务流程的直接映射关系，非常符合人们的认知习惯，特别是从事实际工作的工程技术与管理人员。这也正是离散系统仿真中，基于面向过程建模方法所具有的优势。仿真建模方法中，相邻事件的发生构成的事物称为活动，若干活动组成过程，因此仿真中的过程更接近于流程，即事件发生的顺序。

3.1.5 动态实体与事件

在 GPSS 中，动态实体进入或流出某一模块就意味着某一事件(普通事件或关键事件)的发生或结束(下一临近事件发生)，动态实体的流动路径就是事件发生的顺序。因此，模型的模块图实际上又是实际系统事件和事件发生顺序的描述和概括。如果某一动态实体将在未来某一确定时刻能够流出或流入某一模块，那么，在这一未来的确定时刻所发生的事件必然是关键事件。例如(请参考图 3-1)：

假设模拟开始，时钟当前时间为 0，GENERATE 产生了一个动态实体，根据模块参数(称为操作数)确定到达间隔时间为 14 分钟，计算出动态实体将在未来第 14 分钟流入并流出 GENERATE 而同时流入 QUEUE(队列增加 1 人)模块，意味着在理发馆开门后第 14 分钟时，某顾客到达理发馆一排入队列等待理发的事件一一发生。可以看出顾客到达理发馆为关键事件，而顾客排入队列事件发生在同一时刻，是关键事件引发的普通事件。动态实体继续前进，如果理发师 JOE 闲，则流入 SEIZE(理发师变为忙)—DEPART(队列减少 1 人)—ADVANCE 模块，意味着该顾客得到理发师一离开队列一开始理发的事件，一一发生。此时顾客开始理发的时间即为模拟时钟的当前时间 14，假设由动态实体进入的当前模块 ADVANCE 的参数得到服务持续的时间为 18 分钟，则可计算出该动态实体将在

14＋18＝32 分钟时流出 ADVANCE 同时流入 RELEASE(理发师变为闲)－TERMINATE 模块,这意味将在未来的确定时间,开门后第 32 分钟,该顾客理完发的事件和该顾客让出理发师及顾客离开理发馆的事件发生；同样可以看出：顾客理完发的事件为关键事件,而顾客让出理发师和顾客离开理发馆的事件为普通事件。

以上分析可见关键事件发生时,时钟被修正。实际上,“时钟修正”只发生在以下两种情况中。

(1) 动态实体由 GENERATE 模块产生后通过该模块进入模型时,代表接受服务的实体到达服务系统。

(2) 动态实体进入 ADVANCE 模块滞留一段时间后,代表接受服务的实体接受服务后离开服务实体。

如果以上动态实体并非是第一个到达的,在流出 QUEUE 准备流入 SEIZE 模块时,设备实体 JOE 处于忙态,那么 SEIZE 模块将拒绝该动态实体的进入,意味着顾客获得理发师和离开队列等一系列事件不会发生。于是从模型角度来看,可以认为动态实体仍旧停留在 QUEUE 模块等待。而该动态实体(顾客)得到理发师－离开队列－开始理发等诸事件是否会发生取决于前一动态实体(顾客)理完发这一关键事件是否会发生,因此这三个事件也是普通事件。

动态实体代表实际系统接受服务的实体,它的运动过程就是一系列事件发生的过程,因此它与事件紧密联系而不可分离。一个具体的事件必须由以下三个要素构成。

(1) 事件类型,即发生了什么事,如到达理发馆和排入队列等。

(2) 发生的时间,有两种可能,确定的和不确定的,对于未来某一确定时刻要发生的事件称为关键事件,否则称为普通事件,后一种事件从当前时刻起,随时可能发生,是否发生取决于系统的某些状态条件的出现或存在与否(这些状态条件的出现或存在取决于未来某个关键事件的发生)。

(3) 发生的主体,发生在谁的身上,即哪个动态实体进入队列,表示哪位顾客排入队列的事件发生。

动态实体在处理程序中,被定义为具有复合类型结构的数据体(数组、结构体或类),映射为计算机内存一块连续的存储区,其中就存放有与事件相关的数据,如动态实体编号,每个动态实体的编号必须唯一,以区别其他主体；当前所在模块编号,将要流入的模块编号和未来的移动时间等。

与基于面向事件的建模方法的处理程序相似,GPSS 处理程序也定义了一个事件计划链(称为未来事件链),在这个事件链上按从小到大的未来移动时间顺序,存放会发生关键事件的动态实体。由上分析可见,所有那些将要通过 GENERATE 模块进入模型的动态实体和进入 ADVANCE 模块的动态实体都将被放入这一事件链,并遵守以上排列规则。未来事件链头的第一个动态实体具有最小的未来移动时间。每次时钟的修正就是依据最前端的动态实体所标记的未来移动时间来实现的。时钟修正后,该动态实体被取出,根据其上所标记的路径从当前模块流入下一模块,意味着该动态实体所对应的关键事件发生了,动态实体流经的模块分别实现了对系统状态的修正的操作,这一流动过程一直持续到动态实体停止移动的三个条件之一发生为止。即：

(1) 动态实体进入 ADVANCE 模块,获得未来移动时间后,被重新放到未来事件链。

(2) 动态实体被类似 SEIZE 的模块所拒绝时,停留在其紧前模块。

(3) 动态实体进入 TERMINATE 模块被排出模型。

动态实体沿模型路径的运动和时钟的修正是交替进行的,动态实体的运动过程就是系统状态修正的过程,也是实际系统运行过程的模拟。随着时钟的向前推移,当达到模型事先设定的终止时间时,处理程序便结束模型的运行,并将各种相关统计数据以文件的形式输出。动态实体的运动(系统状态修正)和时钟的修正及它们的交替过程的细节就构成 GPSS 仿真算法的关键部分,实现这一算法的程序模块,也称为 GPSS 调度程序。算法的细节解释将在后面的章节逐步展开。

3.1.6 动态实体的产生与模型输入

动态实体由 generate 模块产生,又通过 generate 模块进入模型。generate 模块每被执行一次,便产生一个动态实体,并为其编号,记录其未来移动时间和下一移入的模块,然后将其放至未来事件链。在以后的某个时钟修正时刻(等于该动态实体的未来移动时间),这一动态实体被送入模型。该动态实体正是通过流入产生它的那个 generate 模块而进入模型的。于是在流入时,这一 generate 模块又被执行一次,而以上的过程又被重复一次,即又由该 generate 模块产生一个新的动态实体,它被标记后,又被放至未来事件链。动态实体就是这样,源源不断地按照一定的间隔时间由某一 generate 模块送入模型。

但问题是:在模型中没有也不允许有流入 generate 模块的路径,那么如何使得某一 generate 模块被执行而产生它的第一个动态实体呢?答案只能有一个,每一个 generate 模块的第一次执行必须由一个外部程序(相对于 generate 模块的内部处理程序)来触发(调用),这个外部程序被称为模型输入操作程序。它执行的操作很简单,即顺序地执行模型的每一个 generate 模块一次,这个过程就称为模型输入操作过程。显然模型输入操作前,时钟时间为 0,未来事件链为空;模型输入操作完成后,未来事件链放置着"N 个"动态实体,它们是由模型中的"N 个"generate 模块各自产生的第一个动态实体,此时模拟时钟仍为 0。

模型输入操作之后,便开始了时钟修正和系统状态修正的交替操作过程,代表模型处于运行状态,图 3-2 说明了以上情况。

3.2 基于 GPSS 的 GPSS/Java

3.2.1 什么是 GPSS/Java

GPSS/Java 是用 Java 语言实现的 GPSS 专用仿真系统。从建模方法和实现的仿真算法来看,它们没有什么区别。

GPSS/Java 提供的建模接口(亦称模块、仿真构件和元素)与 GPSS 几乎一一对应,但形态稍有不同,因为 GPSS/Java 不是用 Java 来实现 GPSS 的一个解释系统或编译系统,而是直接使用 Java 来实现建模接口和仿真算法的程序模块。因此,GPSS/Java 的模型程序是一个使用这些仿真接口所建立的 GPSS 仿真模型的 Java 程序。所有的建模接口和仿真算法的程序模块以类库的形式提供给用户。在设置好类库的路径之后,用户在源程序中只要引

用这一类库，便可使用类库的接口，来建立模型。模型源程序完成后，使用 Java 编译器编译生成的类文件，可由 Java 解释器执行。

GPSS/Java 具有 Java 所具有的一切优点，关于这一方面不再多作论述。

GPSS/Java 实质上向用户提供的只是一个类库而已。但是这种体系结构却具有极大的优越性：

(1) 仿真系统完全向用户开放，用户可以根据需要任意向仿真类库添加实现某种功能的新接口；利用 Java 面向对象的程序设计方法，通过继承或覆盖，可方便地对类库的标准接口做进一步的改造或完善。这就意味着用户不仅是建模者，也可以是系统的开发者，从而将这一用于 GPSS 仿真的类库，发展为具有更为专门化用途的类库，以适用于自己的需要。

(2) GPSS/Java 类库结构高度模块化，模块由 Java 类组成，程序功能的实现是以类为单位来组织的，而所有组成类库的类模块，可以划分为两种，一种称为仿真基础类，一种称为仿真支持类。仿真基础类可划分为动态实体类、资源实体类和操作模块类，直接与 GPSS 模型中的动态实体和资源实体对应，而资源实体类和操作模块类中的接口方法直接映射 GPSS 模型中的模块语句和各种建模所需的标准属性(详见后面章节)。仿真支持类包括仿真调度类、链表操作类、异常处理类、辅助操作符常量定义接口和公用消息类等。其中，仿真调度类封装了仿真调度程序及其启动接口。类库中的许多类实体与仿真模型中的实体一一映射，而模型中的实体又与实际系统中的实体映射，类库的类实体中的接口与模型中的模块语句一一映射，而模型中的模块语句及其顺序又与实际系统的事件和其发生的顺序一一映射。因此 GPSS/Java 实现了从仿真系统到仿真建模再到实际系统空间的直接映射，使得软件结构、软件实现和仿真模型全部统一和构筑在面向对象的基础之上。

(3) 建模语言功能的极大丰富与扩展，由于用户的模型程序首先是一个 Java 程序，所以用户可以在模型程序中使用 Java 的语言部分的所有功能，并且可以使用 Java 本身提供的标准类库或其他开发人员提供的非标准的类库，来实现任何满足构造复杂模型所需要的操作。用户可以直接在模型程序中定义类，调用其中实现的方法或属性。这种允许在模型中调用或嵌入程序的特性化或个性化的处理，使 GPSS/Java 建模具有极大的灵活性，为用户建模提供了更加强大的功能扩展工具。过去建模者只能使用 GPSS 提供的建模构件，或严格遵守某种约定来调用其他有限几种语言函数或过程的接口。

3.2.2　Java 简述

Java 的有关概念已经远远超过了它的语言部分，而日益成为一种软件技术。就是其语言的基础部分，尽管以简洁著称，也很难用几个章节阐述清楚，因此只是将 GPSS/Java 模型程序使用到的 Java 部分语言元素予以说明。

1. 对象与类

现实系统存在的具体事物就是对象，所有性质和行为相似的事物的总称为类。例如某一理发馆的张姓理发师为对象，而所有为顾客理发的人称为理发师，为类。换句话说，某一个提供服务的实体称为服务实体对象，所有的服务实体对象称为服务实体类。

在 Java 程序中，我们要产生某类对象的若干个具体的对象，就要首先定义一个能表述

所有这些对象的类。这个类含有两种属性，一种描述对象的性质或状态称为成员变量，一种描述对象的行为称为成员方法，简称方法。例如：

```
//Facility.java
packge gpssjv;
public class Facility{
    String name;                              //成员变量 1
    int currentState = 0;                     //成员变量 2
    static int count = 1;                     //成员变量 3
    public Facility(){                        //方法 1
        this.name = "FACI" + count;
        count ++ ;
    }
    public Facility(String name){             //方法 2
        this.name = name;
    }
    public void seize(){                      //方法 3
        if(currentState == 0){
            currentState = 1;
            ⋮
            return;
        }
        else{
            ⋮                                 //回到前一模块等待
        }
    }
    ⋮
    public void release(){                    //方法 4
        currentState = 0;
        ⋮
    }
    public int FNU$(){                        //方法 5
        if(currentState == 0)
            return 1;
        else
            return 0;
    }
    ⋮
}
```

其中：关键字 class 表示定义一个类，Facility 表示所定义的类的类名，该段程序定义了一个名为 Facility 的设备实体类。成员变量 1 和变量 2 分别代表设备实体的输出名和当前的忙闲状态，成员变量 3 为静态变量，即全局变量，用作计数器来自动产生设备的输出名。

方法 3 和方法 4 为向用户提供的设备实体的模块语句接口，它们被执行时分别代表设备实体被占用和设备实体被释放。当这两种行为发生时，方法要执行的操作首先是改变设备的当前忙闲状态。方法可以有返回值也可以没有。以上两个方法都只执行某些操作，而没有返回值。方法 5 是具有返回值的方法，它返回设备实体的当前忙闲状态。如果设备在模型中当前处于闲，方法 5 返回值为 1，否则返回 0。

方法又可分为“有参”方法和“无参”方法，例如方法 2 为“有参”方法，其他方法为“无参”方法。在方法被调用时，实参向形参传值，然后执行方法体中的操作。

2. 创建对象

在程序中使用类创建对象，例如：

```
Facility  b1 = new  Facility("Barber Wang");
Facility  b2 = new  Facility("Barber Zhang");
```

b1 和 b2 代表用 Facility 类创建的两个对象，称为引用变量。一个对象在建立时将自动调用构造方法。通常对该对象的某些成员变量的初始化操作安排在构造方法中。构造方法名与类名相同并且没有返回类型。例如以上创建对象 b1 后，立即调用该对象的构造方法 Facility("Barber Wang")，经过实参对形参的值传递，将字符串"Barber Wang"赋给 b1.name。以上模型中创建了两个设备实体对象 b1 和 b2，输出名分别为 Barber Wang 和 Barber Zhang。

一个类中可以定义多个同名的方法，但方法的参数类型或个数须不同，这称为方法重载。构造方法是类中特殊的方法，也可以重载。例如，Facility 类定义了两个构造方法，方法 1 和方法 2，它们重名，但一个有参，一个无参。在创建 Facility 的对象时，可以根据提供的实参个数或类型来选择使用不同的构造方法完成对象的初始化操作。例如：

```
Facility  b3 = new  Facility();
Facility  b4 = new  Facility();
```

以上创建了两个 Facility 类型的对象 b3 和 b4，对应的输出名分别为 FACI1 和 FACI2。

3. this 引用

关键词 this 为一引用变量，通常在某一方法体中使用，代表当前对象。当前对象是指调用某方法的对象，例如当 b1 对象新建后调用构造方法时，该构造方法中的 this 就是 b1，此时，方法 Facility(String name)中

```
this.name = name;
```

等效于

```
b1.name = name;
```

其中，b1.name 为对象 b1 的成员变量，而 name 为构造方法的形参变量。形参变量与在方法内部定义的变量统称为局部变量，它们的作用范围仅限于方法体内部。

类的成员变量和方法的形参变量可以是简单类型，也可以是类类型，如 Facility 中成员变量 name 就是 Java 标准类库中定义的字符串类型的变量，可以存放（或称引用或称代表）一个字符串对象。

4. 继承与覆盖

如果类 A 继承类 B，则称 A 为子类，B 为父类。子类将继承父类中的所有非私有的方法与成员变量。子类的属性由从父类继承的属性和新定义的属性共同组成。Java 只支持单重继承，但允许多层继承。单重继承是指一个子类只能有一个父类，而多层继承是指可以

定义父类的父类。子类将继承其直接或间接父类中的所有非私有属性。Java 使用关键词 extends 来描述两个类的继承关系。例如：

```
class B{
⋮
}
class A extends B{
⋮
}
```

则，A 为 B 的子类。

类的属性具有不同的访问权限，关键词 public 用来表示最大的访问权限，而关键词 private 表示最小的访问权限。程序无法通过对象去访问该对象的私有成员，但是可以访问其公有成员。类具有封装性，对此，可以具有两个层面的理解，一是类的某些属性只能在其对象的内部获得访问，而不能从其对象的外部予以访问，例如用 private 关键字修饰的称为私有成员。类的对象在模型程序中不可能孤立地存在，必然要与其他对象发生联系，而这些与外界发生联系的属性必须能够为外界通过对象的引用去访问，这些属性也称为接口，常定义为公有的成员，即那些用关键字 public 修饰的成员。另一层的理解是：类与对象将数据与对数据进行操作的代码程序封装在一起，数据以成员变量的形式存放，而程序在成员方法中得以实现，从而使得整个程序具有模块化的结构。

如果子类定义的一个方法与其父类中的某一方法的方法名和签名都相同，则称为覆盖。覆盖和被覆盖的方法通常实现不同的操作。

一个“父类”类型的变量不仅可以引用（或称存放或称代表）一个“父类”对象，也可以引用（或称存放或称代表）其子类的对象。如果子类的一个方法覆盖或实现了其父类的方法，那么当我们通过这一存放其子类的对象的“父类”引用变量去调用这个覆盖方法时，调用的将是子类的覆盖方法。相反如果该引用变量存放的是一个“父类”对象，则调用的是父类中的被覆盖方法。当然，其他情况（子类中不存在覆盖其父类的方法）则另当别论。

5. 抽象方法与抽象类

用关键词 abstract 修饰的方法和类，分别称为抽象方法和抽象类；抽象方法只定义方法头，而没有方法的实现，含有抽象方法的类，称为抽象类。

如果一个子类“实现”了其父类的所有抽象方法，则称此子类“实现”了其父类，否则这个子类必须定义为抽象类。例如：

```
abstract  class BlockOp{
  abstract  public void simulate();
}
class Demo1 extends BlockOp{
  public void simulate(){
     ⋮;
  }
}
```