Concurrency

Eko Marpanaji

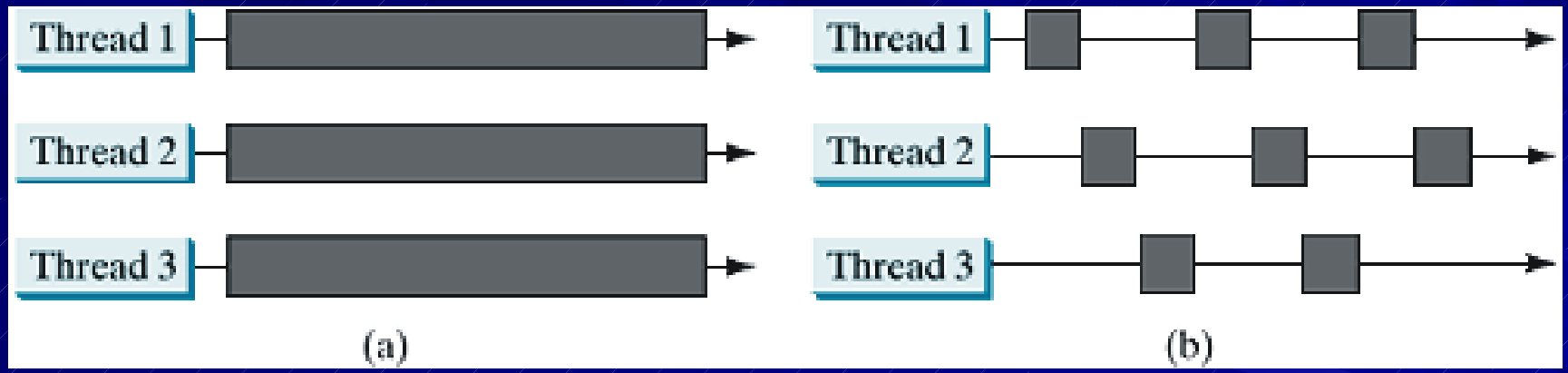
Objectives

- To understand the concept of multithreading and apply it to develop concurrent programs (§24.2).
- To develop task classes by implementing the Runnable interface (§24.3).
- To create threads to run tasks using the Thread class (§24.3).
- To control threads using the methods in the Thread class (§24.4).
- To control animations using threads (§§24.5, 24.7).
- To run code in the event dispatcher thread (§24.6).
- To execute tasks in a thread pool (§24.8).
- To use synchronized methods or blocks to synchronize threads to avoid race conditions (§24.9).
- To synchronize threads using locks (§24.10).
- To facilitate thread communications using conditions on locks (§§24.11–24.12).
- (Optional) To use blocking queues to synchronize access to an array queue, linked queue, and priority queue (§24.13).
- (Optional) To restrict the number of accesses to a shared resource using semaphores (§24.14).
- (Optional) To use the resource-ordering technique to avoid deadlocks (§24.15).
- To understand the life cycle of a thread (§24.16).
- To create synchronized collections using the static methods in the Collections class (§24.17).
- (Optional) To display the completion status of a task using JProgressBar (§24.18).

Thread Concept

- A *thread* is the flow of execution of a task in a program (from beginning to end).
- A task is a program unit that is executed independently of other parts of the program.
- A thread provides the mechanism for running a task. With Java, you can launch multiple threads from a program concurrently. These threads can be executed simultaneously in multiprocessor systems, as shown in Figure 24.1(a).
- In single-processor systems, as shown in Figure 24.1(b), the multiple threads share CPU time, and the operating system is responsible for scheduling and allocating resources to them. This arrangement is practical because most of the time the CPU is idle. It does nothing, for example, while waiting for the user to enter data.

Figure 24.1.
Running Multiple Thread
(a) Multiple CPU (b) Single CPU



Thread Concept (c.)

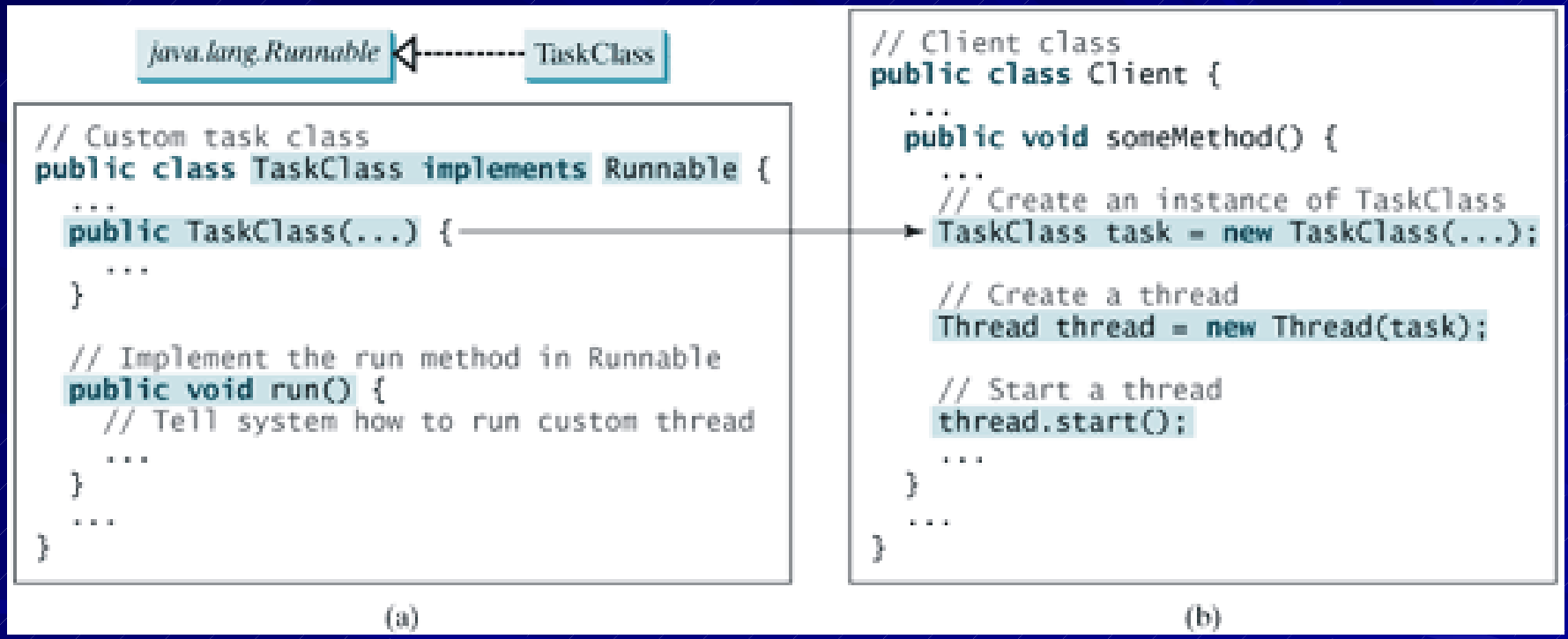
- Multithreading can make your program more responsive and interactive, as well as enhance performance. For example, a good word processor (*misalnya MS Word*) lets you print or save a file while you are typing. In some cases, multithreaded programs run faster than single-threaded programs even on single-processor systems. Java provides exceptionally good support for creating and running threads and for locking resources to prevent conflicts.
- When your program executes as an application, the Java interpreter starts a thread for the main method. When your program executes as an applet, the Web browser starts a thread to run the applet.
- You can create additional threads to run concurrent tasks in the program. In Java, each **task is an instance of the Runnable interface, also called a runnable object.** A thread is essentially an object that facilitates the execution of a task.

Creating Task and Thread

Creating Task

- Tasks are objects.
- To create tasks, you have to first declare a class for tasks.
- A task class must implement the Runnable interface. The Runnable interface is rather simple. All it contains is the run method. You need to implement this method to tell the system how your thread is going to run.
- A template for developing a task class is shown in Figure 24.2(a).

Figure 24.2. Define a Task



- Once you have declared a TaskClass, you can create a task using its constructor. For example,

```
TaskClass task = new TaskClass(...);
```

- A task must be executed in a thread. The Thread class contains the constructors for creating threads and many useful methods for controlling threads. To create a thread for a task, use

```
Thread thread = new Thread(task);
```

- You can then invoke the `start()` method to tell the JVM that the thread is ready to run, as follows:

```
thread.start();
```

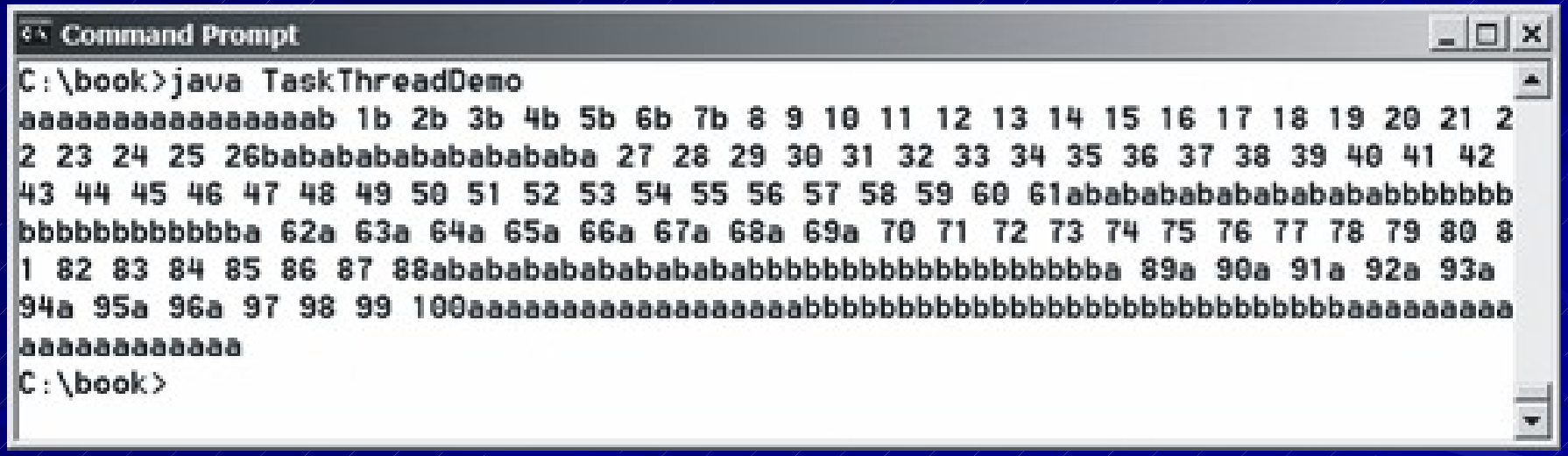
- The JVM will execute the task by invoking the task's `run()` method. Figure 24.2(b) outlines the major steps for creating a task, a thread, and start the thread.

Creating Task and Thread Example

Listing 24.1. A program that creates three tasks and three threads to run them:

- The first task prints the letter a one hundred times.
- The second task prints the letter b one hundred times.
- The third task prints the integers 1 through 100.

Contoh Hasil Eksekusi TaskThreadDemo.java



Importance to note:

- Task adalah *runable object*, sehingga deklarasi kelas untuk task harus `implements Runnable`.
- Thread digunakan untuk menjalankan setiap task
- `run()` adalah method standar yang digunakan oleh Thread untuk menjalankan Task
- `start()` adalah method standar untuk menjalankan Thread