**CPS122 Lecture: State and Activity Diagrams in UML**

last revised February 21, 2022

*Objectives:*

1. To show how to create and read State Diagrams
2. To introduce UML Activity Diagrams

*Materials:*

1. Answers to quick check questions from chapter 7 plus chapter 8 a, b, g
2. Demonstration of "Racers" program
3. Handout and Projectable on Web: State diagram for Session
4. Handout: Code for Session class performSession() method
5. Projectables

## I. Introduction

A. Go over quick check questions chapter 7 + chapter 8 a, b, g only

B. We have drawn a distinction between the static aspects of a system and its dynamic aspects. The static aspects of a system have to do with its component parts and how they are related to one another; the dynamic aspects of a system have to do with how the components interact with one another and/or change state internally over time.

C. We have been looking at one aspect of the dynamic behavior of a system - the way various objects interact to carry out the various use cases. We have looked at two ways of describing this:

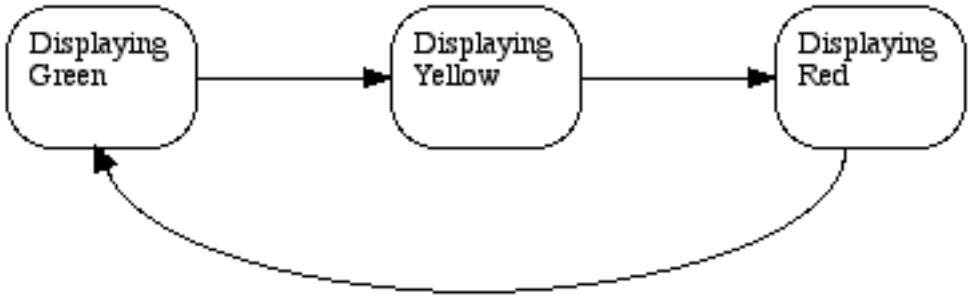
   1. Sequence diagrams

   2. Communication diagrams

D. We now want to look two additional aspects of dynamic behavior

1. How an individual object changes state internally over time.

   a) Example: As part of doing the domain analysis of a traffic light system, it is important to note that individual signals go through a series of states in a prescribed order - modeled by the following state diagram:




   (Note: this is correct for the US but not for all countries in the world!)

   PROJECT

   An important "business rule" that any traffic light system must obey is that the light must be yellow for a certain minimum period of time (related to vehicle speed in the intersection) between displaying green and displaying red. Ensuring that a system exhibits this behavior is essential for safety.

   b) Note the difference between an interaction diagram and a state diagram - the former deals with how several objects work together to accomplish one use case; the latter deals with the internal state of one object, which may be affected by multiple use cases.

   Compare with a person's internal state versus interactions with others. The "class session" use case involves several "Student" objects and a "Professor" object. Something that happens in class may affect the internal state of a student (e.g. getting back a test with a good (or for that matter bad) grade). The internal state of a "Student" object (e.g. sleepy) affects not only this use case, but also others like "chapel".

2. The possibility that several different things may be happening in parallel at the same time.

   Example: Demonstrate “Racers” program.

   Note: the notion of doing things in parallel using threads will be discussed more thoroughly in CPS221

## II. State Diagrams

A. We will now consider a tool that can be used to model this aspect of the behavior of an object: The State Diagram. It can be used either for analysis (if our goal is to record how behavior actually transpires in a system we are modeling) or for design (if our goal is to describe a behavior we want to produce.)

   Note: UML 1 referred to these as statechart diagrams; UML 2 calls them State Machine Diagrams - sometimes shortened to State Diagrams. The book uses the latter term.

B. Often, while we analyze a system, we discover that some objects in a system undergo defined state transitions over time, which is an important part of the dynamic behavior of the system.

   1. Example: we just looked at an example of this in the domain of traffic lights.

   2. What about our ATM example?

      ASK

      a) The ATM itself (represented by an ATM object in our case) goes through a series of states (off, on, serving a customer, etc.)

      b) Individual sessions pass through a series of states

c) Individual transactions also pass through a series of states.

(Note that the objects that have this characteristic are often controller objects.)

C. In such cases, it may prove helpful to develop a State Diagram, showing the states it passes through - especially if there is any complexity to the state transitions.

1. *Example:* We will develop a state diagram for a session.

a) Referring to the use case for a session, what distinct states can we identify?

*ASK*

(1)Reading card

(2)Asking for PIN

(3)Asking for transaction choice

(4)Performing transaction

(5)Ejecting card at the end of the session

(6)Session finished

b) Further, the transitions between states for a session follow a fairly complex pattern - e.g.

(1)After reading a card, we go either to asking for PIN or to ejecting card, depending on whether the card was readable.

(2)After asking for PIN, we normally go to asking for a transaction choice; but we go to ejecting card if the customer presses cancel.

(3)After asking for a transaction choice, we normally go to performing the transaction, but we go to ejecting card if the customer presses cancel.

(4)After performing a transaction, we can go one of three ways:

(a)To choosing transaction, if customer says he/she wants to do another.

(b)To ejecting card, if customer says he/she doesn't want to do another.

(c)To session finished, if the card was retained due to too many invalid PINs.

c) *PROJECT STATE DIAGRAM FOR SESSION FROM WEB*

d) We have already noted that there are other objects for which such a diagram would prove useful.

*ASK*

*SHOW REMAINING STATE DIAGRAMS ON WEB*

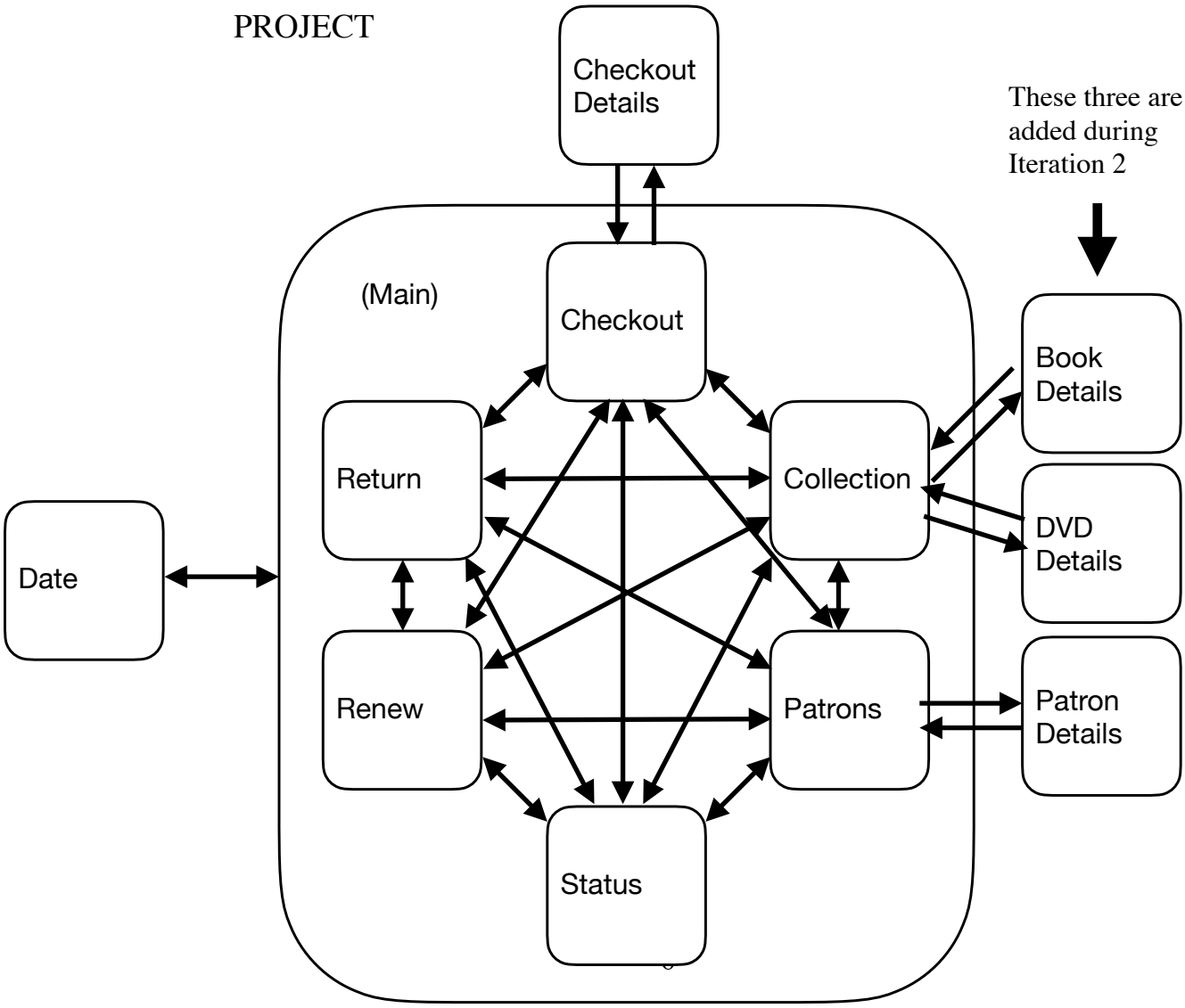
2. In general, a state diagram is a useful analysis tool for:

a) An object that is responsible for a use case, provided that the use case has enough internal complexity to warrant doing a state diagram. (If the use case consists of just a few steps, with no complex variation in sequencing, then a state diagram may not be warranted.)

b) An object has complex internal states, even though it is not directly responsible for a use case

*Example:* In the case of the ATM simulation, the object representing the network connection to the bank would likely warrant a state diagram, because network connections typically pass through a series of states as they dialog with a peer on the other end. We have omitted that from the example, because we've avoided getting into the details of how the network connection works. A real system would, of course, have to deal with this.

D. Another place where state diagrams are often useful is in the design of graphical user interfaces. We will say more about UI design later in the course, but for now here is the GUI for the library project represented as a state diagram. (Each state represents a visible state of the interface)



E. A State diagram makes use of several symbols:

1. A rounded rectangle denotes a state.

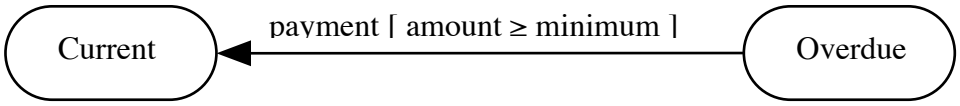
2. Arrows connecting states denote state transitions. A transition is labeled with one or two pieces of information

   a) The situation under which the transition occurs (always)

      (1)This may take the form of a label on the transition itself.

      (2)There may also be a guard associated with the label.

      Example: If we were modeling a ChargeAccount object (representing a credit card account), we might have states "current" and "overdue", with a payment transition from overdue to current guarded by the condition "amount ≥ minimum"




      PROJECT

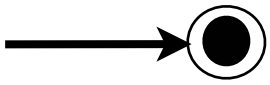
   b) Action that takes place on the transition (optional)

      (in our examples, actions appear on the transitions for the ATM itself, but not for Session and Transaction.)

3. A filled in dot denotes the initial start - where the system starts.

   PROJECT
   (Note: we didn't use this symbol on the GUI State Diagram because it was actually to be understood as part of a larger system)

4. A filled in dot in a circle denotes a final state. When a system reaches the final state, no further state transitions are possible.



PROJECT

(Note that the diagrams for Session and Transaction have final states; but the ATM state diagram does not - turned off is one of its states, but there is no final state per se because it can always be turned back on.)

5. Additional notation is possible, but this is more advanced than what we want to talk about now.

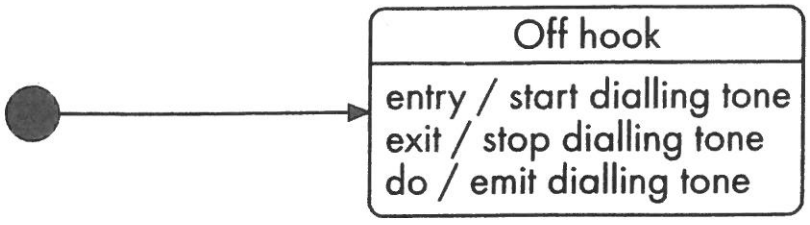
F. It is also possible to incorporate information about specific actions to be performed during a state, or when entering or leaving it.

1. The state diagram for the Overall ATM includes entry actions for two of the three states. These are performed as soon as the state is entered.

2. A state can also have an exit action, to be performed just before leaving the state.

3. A state can have a "do" action, to be performed continuously while in the state.

   Figure 7.12 on page 193 illustrates all of these - though the illustration is a bit forced. Actually, just the "do" action is all that's needed!




PROJECT

G. A state diagram can often be translated into code in a very straightforward way that preserves the clarity of the state diagram

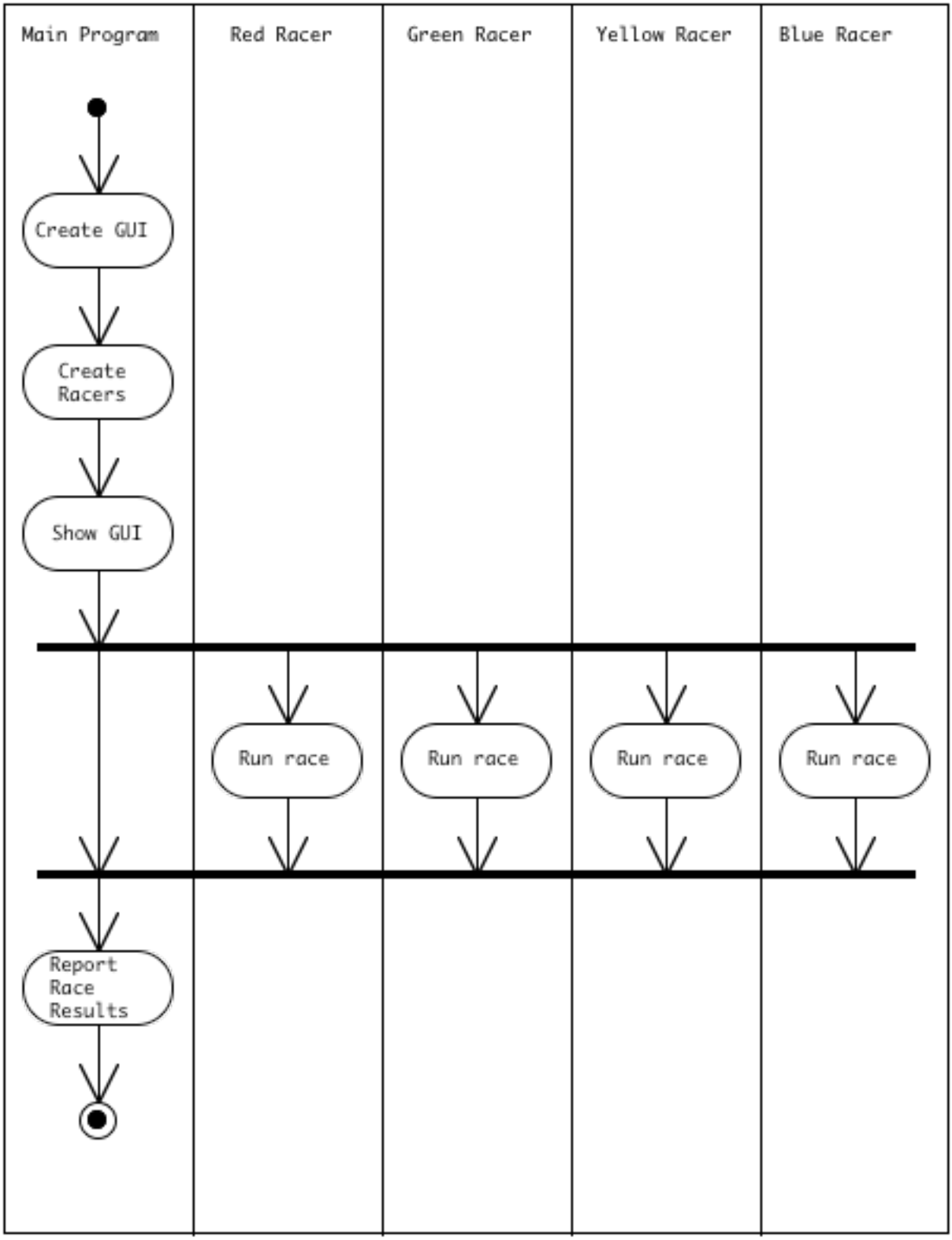
*HANDOUT:* Code for performSession () method of Session - compare to State Diagram on the web.

## III. **UML Activity Diagrams**

A. The other chapter in the book that you read for today discussed a type of UML Diagram called an Activity Diagram. Such a diagram can be constructed for a simple program, but it is particularly useful for showing processes in which there is concurrent processing.

*EXAMPLE:* Demonstrate Racer Program Again

PROJECT Activity Diagram


Main Program
Red Racer
Green Racer
Yellow Racer
Blue Racer
Create GUI
Create Racers
Show GUI
Run race
Run race
Run race
Run race
Report Race Results

B. Features:

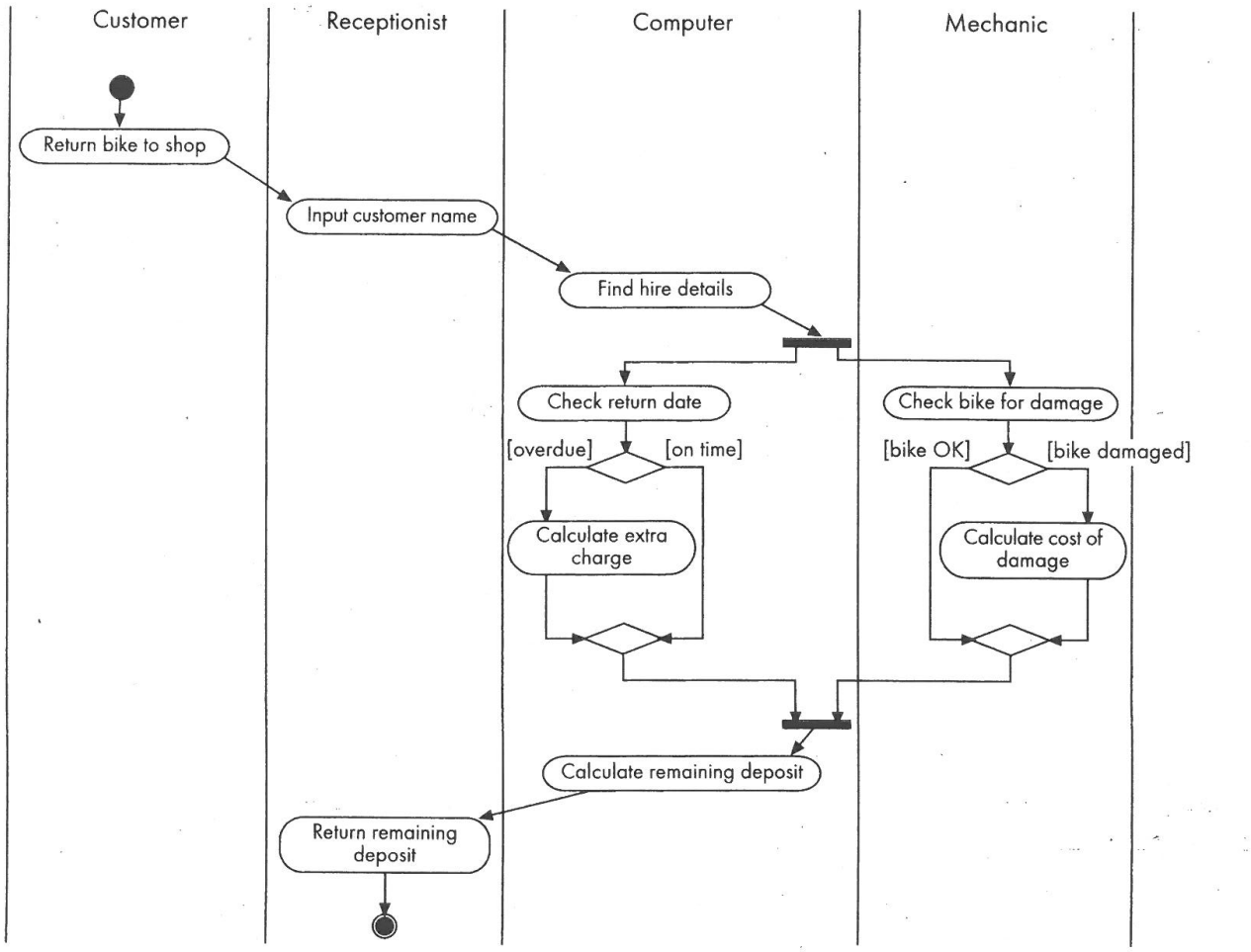
1. Rounded rectangles represent activities.

2. Arrows represent flow from one activity to the next. Each activity is assumed to start as soon as its predecessor completes.

3. Where there is concurrency, the diagram shows "forking" of one thread into two or more, and joining of two or more threads into one. (Forks and joins should match.)

4. The diagram uses "swim lanes" to show the various parts of the system that are performing a task concurrently.

C. *ANOTHER EXAMPLE:* Figure 8.10 on page 209 of the book. In this case the diagram is showing concurrency between three humans and the computer.




PROJECT

D. You will see more about use of concurrency in programs in CPS221.