```
//Author: Shaun Howard, smh150@case.edu
//EECS 338 AS3 - Multiple Sleeping Barbers Problem - Spring 2016

//Initial assumptions about haircuts:
// every thread is paired with a distinct process id
// every thread can access its own process id and will be referenced by
type (Tina, Judy, Dont-Care)
// the barber will store the customer's process id while cutting their
hair
// the customer stores the barber's pid while getting their hair cut
//
//Initial assumptions about environment of operation:
// threads are released by a semaphore in the queuing order
// the maximum line length is 5 people and is denoted as N

//it is assumed there are three separate lines (3 queues), namely Tina-
Only, Judy-Only and Dont-Care-Only,
//and that Dont-Care customers go to their line unless it has more than 5
people, then they will go to either Tina's
//or Judy's line depending on if either has less than 5 people (checking
in that order). Don't Care customers will
//wake up Tina if their line is not full or if their line is full and the
Tina line is not full when they arrive. They will wake up
//Judy only if the Judy line is the only available (non-full) line.
Obviously Tina will be awake if her line is full, so that
//is another assumption. Thus Tina-only customers wake up Tina when they
get there and Judy-only customers wake up Judy
//when they get there.
//Tina or Judy will stay awake if either has at least one customer given
the counting semaphore nature of their operations. Once no customers
//request them any longer, they sleep.
//mod is short for modulo operation.
//int is short for integer data type.
/*
//below are shared variables
//counting semaphores for Tina, Judy
counting semaphore Tina <- 0 //initially sleeping
counting semaphore Judy <- 0 //initially sleeping

//counting semaphores for the three line types
counting semaphore TinaCusts <- 0  //initially closed (barber has to wake
up to serve customers)
counting semaphore JudyCusts <- 0
counting semaphore DontCareCusts <- 0

//integer customer limit per line
int N <- 5

// please note that time_stamp is a data type that acts like a float and
the larger it is, the more recent
// the time is (in order to maintain FCFS across the barber shop)
int TinaSeats[N] <- empty tuple (integer, time_stamp) array of size N
int JudySeats[N] <- empty tuple (integer, time_stamp) array of size N
int DontCareSeats[N] <- empty tuple (integer, time_stamp) array of size N

//track the free seats for the Tina line
int TinaFreeSeats <- N

//index of the next customer seat as a rolling sum that resets using mod
N
```

```
int TinaCustNextSeat <- 0

//track Tina's next customer
int TinaNextServe <- 0

//leave Tina seat access open
binary semaphore TinaSeatAccess <- 1

//track free seats of Judy line
int JudyFreeSeats <- N

//index for next Judy cust seat, resets with mod N
int JudyCustNextSeat <- 0

//track Judy's next customer
int JudyNextServe <- 0

//initially open Judy seat access
binary semaphore JudySeatAccess <- 1

//dont-care versions of above
int DontCareFreeSeats <- N
int DontCareCustNextSeat <- 0
binary semaphore DontCareSeatAccess <- 1

//track next dont-care customer in line
int DontCareNextServe <- 0
*/


/**
first solution: Tina as Barber
the thread for Tina must be started by the main program before any others
Tina will try to sleep if she has no customers.
Tina {
    while(true){
        //sleep until awoke by customer
        wait(Tina)

        //protect seat changes in Tina line
        wait(TinaSeatAccess)

        //protect seat changes in Dont-Care line
        wait(DontCareSeatAccess)

        //peek at the next Tina customer
        integer next_tina_cust <- TinaNextServe mod N

        //peek at the next Dont-care customer
        integer next_dontcare_cust < DontCareNextServe mod N

        if (TinaFreeSeats < N and DontCareFreeSeats < N) {
           //check if the next Tina customer has an older time_stamp than
the dont care next customer
           if (TinaSeats[next_tina_cust] <=
DontCareSeats[next_dontcare_cust]){
                //unlock dont-care seats
                signal(DontCareSeatAccess)

                //do the Tina barber operation
                doTina()
            else {
                //unlock Tina seats
                signal(TinaSeatAccess)

                //do the Don't Care barber operation
                doDontCare()
            }
        } else if (TinaFreeSeats < N) {
            //unlock dont-care seats
            signal(DontCareSeatAccess)
            doTina()
        } else if (DontCareFreeSeats < N) {
            signal(TinaSeatAccess)
            doDontCare()
        } else {
            //unlock seat access
            signal(TinaSeatAccess)
            signal(DontCareSeatAccess)
        }
    }
}
 */


/**
second solution: Judy as Barber
the thread for Judy must be started by the main program after Tina.
Judy will try to sleep if she has no customers.
Judy {
    while(true){
      //sleep until awoke by customer
        wait(Judy)

        //protect seat changes in Judy line
        wait(JudySeatAccess)

        //protect seat changes in Dont-Care line
        wait(DontCareSeatAccess)

        //peek at the next Judy customer
        integer next_judy_cust <- JudyNextServe mod N

        //peek at the next Dont-care customer
        integer next_dontcare_cust < DontCareNextServe mod N

        if (JudyFreeSeats < N and DontCareFreeSeats < N) {
           //check if the next Judy customer has an older time_stamp than
the dont care next customer
           if (JudySeats[next_judy_cust] <=
DontCareSeats[next_dontcare_cust]){
                //unlock dont-care seats
                signal(DontCareSeatAccess)

                //do the Judy barber operation
                doJudy()
           else {
                //unlock Judy seats
                signal(JudySeatAccess)
                //do the Don't Care barber operation
                doDontCare()
            }
        } else if (JudyFreeSeats < N) {
            //unlock dont-care seats
            signal(DontCareSeatAccess)
            doJudy()
        } else if (DontCareFreeSeats < N) {
            signal(JudySeatAccess)
            doDontCare()
        } else {
            //unlock seat access
            signal(JudySeatAccess)
            signal(DontCareSeatAccess)
        }
    }
}
 */


/**
third solution: Tina-only customers
threads for Tina's Customers should be started at random times by main
program for continuous customer approach
TinaCustomer {
    //protect seat changes with mutex
    wait(TinaSeatAccess)

    //stay if there are less than 5 people (up to 5 free seats total)
    if (TinaFreeSeats > 0) {
        doTinaCustomer()
    } else {
        //free the seat change mutex
        signal(TinaSeatAccess)
    }
    //customer either had hair cut or left
    exit current thread or process
}
 */


/**
fourth solution: Judy-only customers
threads for Judy's Customers should be started at random times by main
program for continuous customer approach
JudyCustomer {
    //protect seat changes with mutex
    wait(JudySeatAccess)

    //stay if there are less than 5 people (up to 5 free seats total)
    if (JudyFreeSeats > 0) {
        doJudyCustomer()
    } else {
        //free the seat change mutex
        signal(JudySeatAccess)
    }
    //customer either had hair cut or left
    exit current thread or process
}
 */
```

```
/**
fifth solution: Don't-Care customers
threads for Don't-Care Customers should be started at random times for
continuous customer approach by main program
a dont-care customer first checks the dont-care line, then the Tina line,
then the Judy line and then leaves
DontCareCustomer {
    //attempt to access don't-care line
    wait(DontCareSeatAccess)
    if (DontCareFreeSeats > 0) {
        doDontCareCustomer()
    } else {
        signal(DontCareSeatAccess)
    }

    //attempt to access Tina line
    wait(TinaSeatAccess)
    if (TinaFreeSeats > 0) {
        doTinaCustomer()
    }  else {
        signal(TinaSeatAccess)
    }

    //attempt to access Judy line
    wait(JudySeatAccess)
    if (JudyFreeSeats > 0) {
        doJudyCustomer()
    } else {
        signal(JudySeatAccess)
    }
    exit current thread or process
}
 */


/**
The Tina operation when Tina cuts hair, signals TinaSeatAccess and
TinaCusts when done
void doTina(){
    //get next customer and reset value based on modulo operation (make
circular buffer)
    TinaNextServe <- TinaNextServe mod N
    int next_cust <- TinaNextServe

    //increment Tina's next customer
    TinaNextServe++

    //find the customer pid
    int cust_pid <- TinaSeats[next_cust]

    //leave customer Tina's pid
    TinaSeats[next_cust] <- Tina's process id

    //allow seat changes by releasing mutex
    signal(TinaSeatAccess)

    //wake up customer
    signal(TinaCusts)
    //Tina cuts the hair of customer C
    cut_hair("Tina")
}
*/


/**
The Judy operation when Judy cuts hair, signals JudySeatAccess and
JudyCusts when done
void doJudy(){
    //get next customer and reset value based on modulo operation (make
circular buffer)
    JudyNextServe <- JudyNextServe mod N
    int next_cust <- JudyNextServe

    //increment Judy's next customer
    JudyNextServe++

    //find the customer pid
    int cust_pid <- JudySeats[next_cust]

    //leave customer Judy's pid
    JudySeats[next_cust] <- Judy's process id

    //allow seat changes by releasing mutex
    signal(JudySeatAccess)

    //wake up customer
    signal(JudyCusts)

    //Judy cuts the hair of customer C
    cut_hair("Judy")
}
 */


/**
The Dont-Care barber operation when whoever cuts hair, signals
DontCareSeatAccess and DontCareCusts when done
void doDontCare(){
    //get next customer and reset value based on modulo operation (make
circular buffer)
    DontCareNextServe <- DontCareNextServe mod N
    int next_cust <- DontCareNextServe

    //increment DontCare's next customer
    DontCareNextServe++

    //find the customer pid
    int cust_pid <- DontCareSeats[next_cust]

    //leave customer DontCare's pid
    DontCareSeats[next_cust] <- DontCare's process id

    //allow seat changes by releasing mutex
    signal(DontCareSeatAccess)

    //wake up customer
    signal(DontCareCusts)
    //DontCare cuts the hair of customer C
    cut_hair("DontCare")
}
 */


/**
The Tina line routine to execute when there are Tina-only seats open.
A customer has Tina seat access, sits down, signals seat access, and then
signals Tina,
then waits in the Tina line until signaled, then they get a haircut from
Tina and leave.
void doTinaCustomer(){
    //take a seat in the Tina line
    TinaFreeSeats <- TinaFreeSeats - 1

    //increment the customer seat position
    TinaCustNextSeat <- TinaCustNextSeat mod N

    //chose the new seat position
    int cust_seat <- TinaCustNextSeat

      //increment the customer seat id
      TinaCustNextSeat++

    //store the customer's pid
    TinaSeats[cust_seat] <- customer's process id

    //free the seat change mutex
    signal(TinaSeatAccess)

    //wake up Tina
    signal(Tina)

    //wait in the Tina line
    wait(TinaCusts)
    wait(TinaSeatAccess)

    //store Tina's pid
    int barber_pid <- TinaSeats[cust_seat]

    //allow seat to be used for more customers
    TinaFreeSeats <- TinaFreeSeats + 1
    signal(TinaSeatAccess)

    //get haircut from Tina
    get_haircut("Tina)
}
/


/**
The Judy line routine to execute when there are Judy-only seats open.
void doJudyCustomer(){
    //take a seat in the Judy line
    JudyFreeSeats <- JudyFreeSeats - 1

    //increment the customer seat position
    JudyCustNextSeat <- JudyCustNextSeat mod N
    //chose the new seat position
    int cust_seat <- JudyCustNextSeat

      //increment the customer seat id
      JudyCustNextSeat++

    //store the customer's pid
    JudySeats[cust_seat] <- customer's process id

    //free the seat change mutex
    signal(JudySeatAccess)

    //wake up Judy
    signal(Judy)

    //wait in the Judy line
    wait(JudyCusts)
    wait(JudySeatAccess)

    //store Judy's pid
    int barber_pid <- JudySeats[cust_seat]

    //allow seat to be used for more customers
    JudyFreeSeats <- JudyFreeSeats + 1
    signal(JudySeatAccess)

    //customer gets hair cut by Judy
    get_haircut("Judy")
}
 */


/**
The don't-care line routine to execute when there are dont-care seats
open.
void doDontCareCustomer(){
    //take a seat in don't care line
    DontCareSeats <- DontCareSeats - 1

    //get the current customers index
    DontCareCustNextSeat <- DontCareCustNextSeat mod N

    //store the seat index
    int cust_seat <- DontCareCustNextSeat

    //increment the next seat position
    DontCareCustNextSeat++

    //store the customer's process id at that index seat
    DontCareSeats[cust_seat] <- customer's process id

    //give back seat access
    signal(DontCareSeatAccess)

    //wake up Tina first (shop owner)
    signal(Tina)

    //wait in the dont care custs line
    wait(DontCareCusts)
    wait(DontCareSeatAccess)

    //retrieve the barber's pid
    int barber_pid <- DontCareSeats[cust_seat]

    //increment the free seat count
    DontCareSeats <- DontCareSeats + 1

    //signal an open dont-care seat
    signal(DontCareSeatAccess)

    //get haircut from either Tina or Judy
    get_haircut("DontCare")
}
 */

/**
The cut hair function that will make the given barber who is cutting hair
busy.
It takes the barber name as a string or "DontCare" if servicing a don't
care customer.
It will handle the necessary sleeping or cutting process that will be
communicated
with by the get_haircut function to successfully coordinate completion
and termination
of the hair cut session.

void cut_hair(string barber)
 */

/**
The get hair cut function that will make the given customer who is
getting their hair
cut busy. It will take a string of the line type the customer is from to
determine
how to interact with the barber when they are cutting the customer's
hair. It will
be communicated with by the cut_hair function in order to enable
successful completion
and termination of the hair cut session.

void get_haircut(string customer_type)
 */
```