Homework #6

Homework Title: Thread Synchronization and Cooperation

Student ID: 201724542 Name: 이준영

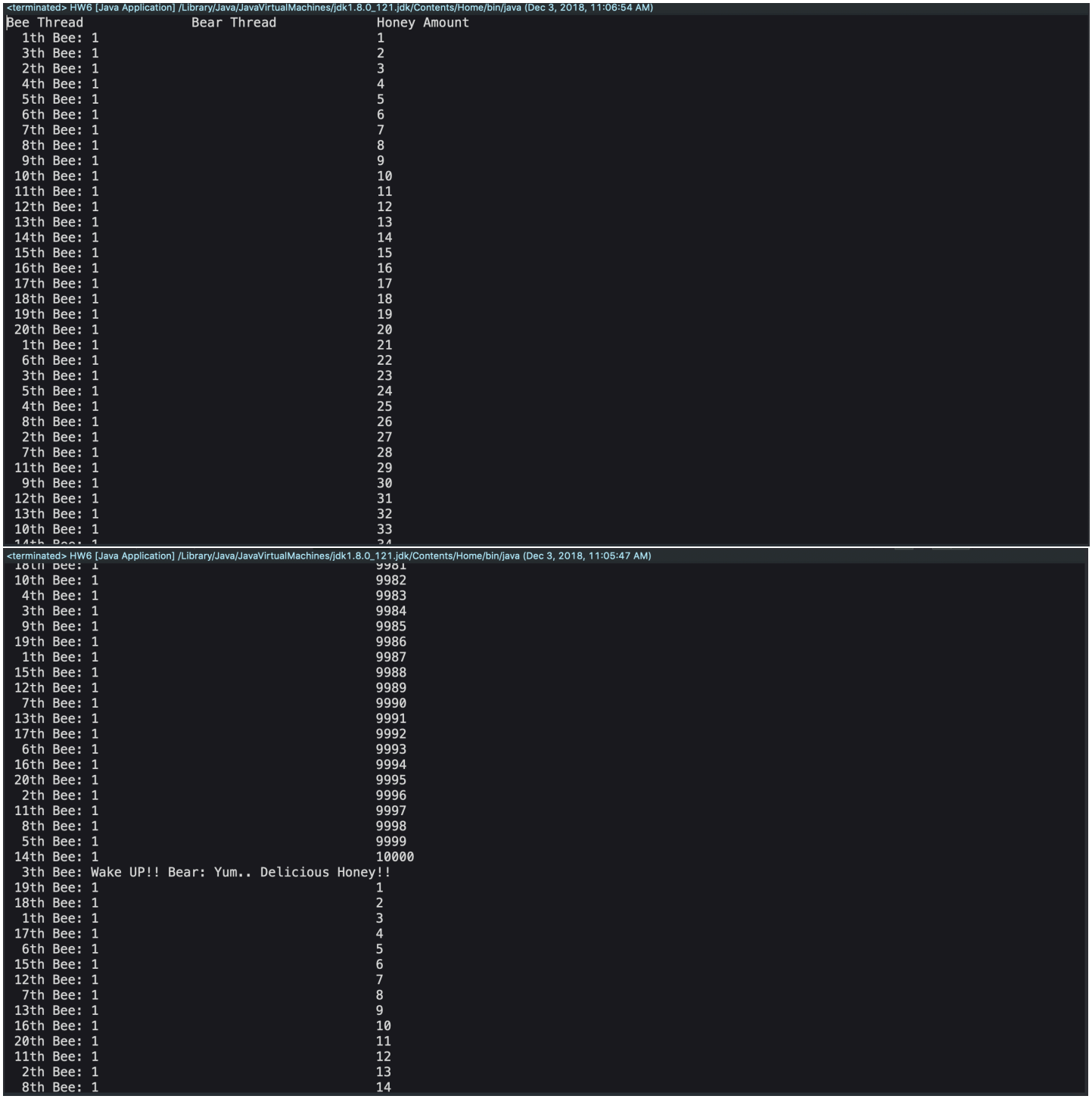
(Problem 1) Use lock/unlock and signal/await for thread's synchronization and cooperation.

1) Describe the program source with detailed comments.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class HW6 { // 메인 클래스
	private static Pot honeyPot = new Pot(); // Pot 클래스 변수 생성

	public static void main(String[] args) { // 메인 함수
		System.out.println("Bee Thread\t\tBear Thread\t\tHoney Amount");
		// 출력 포맷을 위한 문장 출력

		ExecutorService executor = Executors.newFixedThreadPool(21);
		// Bee 개수 + Bear 개수 == 21 이므로 그에 맞는 쓰레드 풀 생성

		for (int i = 1; i <= 20; i++) { // 반복문으로 Bee 쓰레드를 쓰레드 풀에 추가
			executor.execute(new Bee(i));
		}

		executor.execute(new Bear()); // Bear 쓰레드를 쓰레드 풀에 추가
	}

	private static class Bee implements Runnable { // Bee 클래스
		int beeNum; // 몇번째로 생성된 벌인지를 나타내는 int 변수

		Bee(int value) { // 생성자 재선언
			beeNum = value; // 입력값을 beeNum 에 저장
		}

		@Override
		public void run() { // run() 함수를 오버라이딩
			try {
				int honey = 1; // 저장할 꿀의 양을 1 로 설정
				while(true) {
					honeyPot.add(honey, beeNum); // 꿀단지에 꿀을 추가하는 메소드 실행
					Thread.sleep(((int) Math.random() * 10000) % 1900 + 101);
					// Bee 쓰레드를 100 ~ 2000ms 동안 중지
				}
			} catch (InterruptedException e){
				e.printStackTrace();
			}
		}
	}

	private static class Bear implements Runnable { // Bear 클래스

		@Override
		public void run() { // run() 함수를 오버라이딩
			while(true) {
```

```
                        honeyPot.eat(); // 꿀을 다 먹어버리는 메소드 실행
                }
        }
}

private static class Pot { // Pot 클래스
        private static final int H = 10000; // 꿀단지의 용량을 나타내는 int 변수
        private int honeyAmount = 0; // 저장된 꿀의 양을 나타내는 int 변수
        private int lastBee; // 마지막으로 꿀을 넣은 벌을 나타내는 int 변수

        private static Lock Lock = new ReentrantLock(); // Lock 변수 생성
        private static Condition Empty = Lock.newCondition();
        // 꿀단지가 비었다는 상태를 나타낼 Condition 변수 생성

        private static Condition Full = Lock.newCondition();
        // 꿀단지가 다 찼다는 상태를 나타낼 Condition 변수 생성

        public void add(int honey, int beeNum) { // 꿀단지에 꿀을 추가하는 메소드
                Lock.lock(); // Lock 을 잠금
                try {
                        lastBee = beeNum; // 마지막으로 꿀을 넣은 벌을 저장
                        if(honeyAmount == H) { // 만약 현재 꿀의 양이 10000 과 같으면
                                Full.signal();
                                // 다 찼다고 신호를 보냄
                                Empty.await();
                                // 곰이 다 먹고 단지가 비었다는 신호가 올 때까지 대기
                        }
                        honeyAmount += honey; // 꿀단지에 꿀을 추가
                        System.out.println(String.format("%12s",beeNum + "th Bee: " + honey) +
                                        "\t\t\t\t\t" + honeyAmount);
                        // 몇번째 벌이 얼마만큼 꿀을 추가했고, 그 후 잔여량이 얼마인지를 모두 출력
                }  catch (InterruptedException e){
                        e.printStackTrace();
                }
                finally {
                        Lock.unlock(); // Lock 을 다시 해제
                }
        }

        public void eat() { // 꿀단지에서 곰이 꿀을 먹는 메소드
                Lock.lock(); // Lock 을 잠금
                try {
                        while(honeyAmount != H) { // 만약 현재 꿀의 양이 10000 이 아니면
                                Full.await(); // 꿀단지에 꿀이 다 찰 때까지 대기
                        }
                        System.out.println(String.format("%20s",lastBee + "th Bee: Wake UP!!") +
                                        " Bear: Yum.. Delicious Honey!!");
                        // 꿀을 다 먹었다고 출력
                        honeyAmount = 0; // 꿀단지를 모두 비움
                        Empty.signal(); // 꿀단지가 비었다고 신호를 줌
                }catch (InterruptedException e){
                        e.printStackTrace();
                }
                finally {
                        Lock.unlock(); // Lock 을 해제
                }

        }
}
}
```

2) Capture the outcome generated by your program on the screen.



(Problem 2) Use lock/unlock and notify/wait for thread's synchronization and cooperation.

1) Describe the program source with detailed comments.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class HW6_2 { // 메인 클래스
	private static Pot honeyPot = new Pot(); // Pot 클래스 변수 생성

	public static void main(String[] args) { // 메인 함수
		System.out.println("Bee Thread\t\tBear Thread\t\tHoney Amount");
		// 포맷에 맞춘 출력을 위한 문장 출력

		ExecutorService executor = Executors.newFixedThreadPool(21);
		// 쓰레드 풀 생성

		for (int i = 1; i <= 20; i++) { // 반복문으로 쓰레드 풀에 Bee 쓰레드 추가
			executor.execute(new Bee(i));
		}

		executor.execute(new Bear()); // 쓰레드 풀에 Bear 쓰레드 추가
		executor.shutdown();
	}

	public static class Bee implements Runnable { // Bee 클래스
		private int beeNum; // 벌의 번호를 저장할 int 변수

		Bee(int value) { // 생성자 재선언
			beeNum = value; // 입력값을 beeNum에 저장
		}

		@Override
		public void run() { // run() 함수를 오버라이딩
			try {
				while(true) {
					synchronized(honeyPot) { // Pot 쓰레드를 동기화
						Pot.lastBee = beeNum; // 가장 마지막에 넣은 벌의 번호를 저장
						if (honeyPot.getHoneyAmount() != Pot.H) { // 만약 현재 꿀의
양이 10000이 아니면
							honeyPot.add(1, beeNum); // 꿀을 1만큼 더하는 메소드를
실행
							honeyPot.notify(); // Pot 쓰레드에 알림을 준다
						}
					}
					Thread.sleep(((int) Math.random() * 10000) % 1900 + 101); // 100 -
2000ms 동안 쓰레드를 재운다
				}
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}

	public static class Bear implements Runnable { // Bear 클래스
		@Override
		public void run() { // run() 함수를 오버라이딩
			try {
				while (true) {
					synchronized(honeyPot) { // Pot 쓰레드를 동기화
```

```
                                        while(honeyPot.getHoneyAmount() < Pot.H) // 꿀단지가 가득 차지
않았을 동안
                                                honeyPot.wait(); // Pot 쓰레드의 notify를 기다림
                                        honeyPot.eat(); // 꿀을 먹는 메소드를 실행
                                }
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
    }

    private static class Pot { // Pot 클래스

        private static final int H = 10000; // 꿀단지의 용량을 나타내는 int 변수
        private int honeyAmount = 0; // 현재 담긴 꿀의 양을 나타내는 int 변수
        private static int lastBee; // 마지막에 꿀을 넣은 벌을 나타내는 int 변수

        public int getHoneyAmount() { // 현재 담긴 꿀의 양을 반환하는 함수
                return honeyAmount; // honeyAmount를 반환
        }

        public void eat() { // 꿀을 먹는 메소드
                honeyAmount = 0; // 꿀의 양을 0으로 바꿈
                System.out.println(String.format("%20s",lastBee + "th Bee: Wake UP!!") +
                                " Bear: Yum.. Delicious Honey!!");
                // 포맷에 맞게 어떤 벌이 곰을 깨웠는지와 곰이 꿀을 먹었다는 내용을 출력
        }

        public void add(int honey, int beeNum) { // 꿀을 추가하는 메소드
                if (honey+honeyAmount <= H) honeyAmount += honey; // 꿀을 추가
                System.out.println(String.format("%12s",beeNum + "th Bee: " + honey) +
                                "\t\t\t\t\t" + getHoneyAmount());
                // 포맷에 맞게 어떤 벌이 꿀을 얼마만큼 추가했는지를 출력
        }
    }
}
```

2) Capture the outcome generated by your program on the screen.