Code Walkthrough

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

Readings and References

- Reading
  - » Chapter 6, *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch
    - The code in this lecture is an extended version of the example given in Chapter 6

```
/**
 * This class uses a Cracker to open a DigitalLock.
 */
public class WhiteHat {
  /**
   * Create various locks and try to pick them.
   * @param arg user arguments.  Ignored in this implementation
   */
  public static void main(String arg[]) {
    DigitalLock lock;
    int combo;
    Cracker fingers = new Cracker();
    for (int i=0; i<=1000; i++) {
      lock = new DigitalLock(i);
      combo = fingers.pick(lock);
      if (combo != i) {
        System.out.println("Couldn't pick "+i+", returned "+combo);
      }
    }

    // create one more lock with a random combination

    int theCombination = (int)(Math.random()*1000);
    lock = new DigitalLock(theCombination);
    System.out.println("Random combination was "+fingers.pick(lock));
  }
}
```

WhiteHat Ponder points

- How many local variables of type DigitalLock are declared in the main method?
- How many objects of type DigitalLock are created when WhiteHat main(…) runs?
- How many objects of type Cracker are created when WhiteHat main(…) runs?
- Are any parameters passed to the Cracker constructor?
- Are any parameters passed to the pick method of the Cracker object?
- Can we tell from looking at main(…) what the return type from pick(lock) is?
- Can we tell from looking at main(…) what the return type from Math.random() is?
- How would you start this program running? Where does execution first begin?

```
/**
 * This class is used in conjunction with a lock class to see if it
 * can open the lock without knowing the original combination.
 */
public class Cracker {
  /**
   * Attempt to pick the given lock by guessing the combination.
   * @param lock the lock to work on
   * @return the combination that worked or -1 if failed.
   */
  public int pick(DigitalLock lock) {
    for (int i = 0; i<10; i++) {
      for (int j=0; j<10; j++) {
        for (int k=0; k<10; k++) {
          lock.close();
          lock.enter(i);
          lock.enter(j);
          lock.enter(k);
          if (lock.isOpen()) {
            return i*100+j*10+k;
          }
        }
      }
    }
    return -1;
  }
}
```

Cracker Ponder Points

- Is there a constructor for Cracker?
- What type of parameter is passed to the pick method?
- Are there any local variables used in the pick method?
- How many times does the pick method call the isOpen method if the combination is 99?
- Under what circumstances will pick return 0 to its caller?
- Under what circumstances will pick return -1 to its caller?

```
/**
 * A lock with a three digit combination.  This class is derived from
 * the example in Nino and Hosch, chapter 6.  These locks are created in the
 * open state.  They can be closed, then reopened by providing the right
 * combination of digits.  Also, these locks are welded closed after certain
 * errors.  Once a lock is welded, no entry will match the combination
 * and the lock stays closed forever.
 */
public class DigitalLock {

  /** state of the lock */
  private boolean open;

  /** certain errors will cause the lock to be welded after which it won't open */
  private boolean welded;

  /** first (leftmost) digit of the combination.  0 to 9 */
  private int comb1;
  /** second (middle) digit of the combination.  0 to 9 */
  private int comb2;
  /** third (rightmost) digit of the combination.  0 to 9 */
  private int comb3;

  /**
   * The first of the last three digit entries. -1 indicates the digit
   * has not been entered.
   */
  private int entered1;
  /** The second of the last three digit entries. */
  private int entered2;
  /** The last of the last three digit entries (ie, the most recent entry). */
  private int entered3;
```

DigitalLock instance variables

DigitalLock constructor

```
/**
  * Create a lock with the given three digit combination.  The lock
  * is in the open state after it is created unless an invalid
  * combination setting is supplied in which case the lock is welded
  * closed and can never be opened.
  * Combination values < 100 are assumed to have leading zeros.
  * @param theCombination the combination value for this lock.
  * The digit in the 100's position is considered to be the first
  * digit of the combination, the 10's position is the second digit,
  * and the units position is the third digit.
  * This value must be >=0 and <=999.  If it is not, then
  * the lock is welded closed.
  */
 public DigitalLock(int theCombination) {
   setCombination(theCombination);
   clearEntries();
 }
```

```
/**
  * Enter a digit of the combination.  The lock is opened if
  * the three digits of the combination are entered in
  * order.
  * @param digit the single digit entry.  Must be a single decimal digit 0-9.
  */
 public void enter (int digit) {
   if (digit >= 0 && digit <= 9 ) {
     entered1 = entered2;
     entered2 = entered3;
     entered3 = digit;
     if (isValidCombination(entered1, entered2, entered3)) {
       open = true;
     }
   }
 }
 /**
  * Get the state of the lock.
  * @return true if the lock is open, false if the lock is closed.
  */
 public boolean isOpen() {
   return !welded && open;
 }
 /**
  * Close this lock.  Any partially entered combination is cleared.
  */
 public void close() {
   open = false;
   clearEntries();
 }
```

Some public methods

A private method

```
/**
  * Set the combination of the lock to a new value.  The lock
  * is in the open state after this operation unless an invalid
  * combination is supplied in which case the lock is welded closed
  * and can never be opened.
  * Combination values < 100 are assumed to have leading zeros.
  * @param theCombination the combination value for this lock.
  * The digit in the 100's position is considered to be the first
  * digit of the combination, the 10's position is the second digit,
  * and the units position is the third digit.
  * This value must be >=0 and <=999.  If it is not, then
  * the lock is welded closed.
  */
 private void setCombination(int theCombination) {
   if (theCombination >=0 && theCombination <=999) {
     welded = false;                 // this lock can open and close
     comb1 = (theCombination / 100) % 10;
     comb2 = (theCombination / 10) % 10;
     comb3 = (theCombination) % 10;
   } else {
     welded = true;                  // this lock will never open
     comb1 = comb2 = comb3 = 0;      // doesn't matter, lock is welded
   }
   open = !welded;
 }
```

```
  /**
    * Check to see if the given value is a valid combination for this lock.
    * @param theC the combination to check
    * @return true if the proposed combination will unlock the lock.
    */
   private boolean isValidCombination(int theC) {
     return isValidCombination((theC/100)%10,(theC/10)%10,(theC)%10);
   }
   /**
    * Check to see if the given values form a valid combination for this lock.
    * If the lock is welded closed, then no proposed combination will be
    * accepted as valid, even if it does match the lock's combination.
    * @param first the first digit of the proposed combination
    * @param second the second digit of the proposed combination
    * @param third the third digit of the proposed combination
    * @return true if the proposed combination will unlock the lock.  Note that
    * if the lock is welded closed, then nothing will unlock it.
    */
   private boolean isValidCombination(int first, int second, int third) {
     if (welded) return false;
     return (first == comb1) && (second == comb2) && (third == comb3);
   }
   /**
    * Clear previous entries.
    */
   private void clearEntries() {
     entered1 = -1;
     entered2 = -1;
     entered3 = -1;
   }
}  // end of class definition
```

More private methods

DigitalLock Ponder Points

- Are any local variables used in the DigitalLock constructor?
- Does the enter(int digit) method return a value to its caller?
- Does the isOpen() method return a value to its caller?
- What does the enter(int digit) method do if the given digit is not valid? What if it is valid?
- How does isValidCombination(int theC) calculate the value that it returns to its caller?