```java
/**
 * ISPCharge represents an internet charge
 *
 * @author Nancy Harris
 * @version V1 10/2013
 */
public class ISPCharge
{
	private final double A_CEILING = 10.0;
	private final double A_PRICE_HOUR = 2.0;
	private final double A_PRICE_MONTH = 9.95;
	private final double B_CEILING = 20.0;
	private final double B_PRICE_HOUR = 1.0;
	private final double B_PRICE_MONTH = 13.95;
	private final double C_PRICE_MONTH = 19.95;
	private final double TAX_RATE = .05;

	// variables describing this charge.
	private char packageCode;
	private double hours;

	/*****************************************************
	 * The constructor sets the package and hours attributes. This assumes that
	 * the pkg code is correct
	 *
	 * @param pkg
	 *            The code for the package, A, B, or C
	 * @param hours
	 *            The number of hours this month
	 */
	public ISPCharge(char pkg, double hrs)
	{
		this.packageCode = pkg;
		this.hours = hrs;
	}

	/*************************************************
	 * calc charge will decide which package to apply and will return the
	 * correct cost.
	 *
	 * @return The charges for this month.
	 */
	public double calcCost()
	{
		double cost;

		switch (packageCode)
		{
		case 'A':
		case 'a':
			cost = calcA();
			break;
		case 'B':
		case 'b':
			cost = calcB();
			break;
		case 'C':
		case 'c':
			cost = calcC();
			break;
		default:
			cost = 0;
		}
		return cost;
```



```java
	}

	/*************************************************
	 * calcA calculates the charges for package A
	 *
	 * @return The cost for package A
	 */
	public double calcA()
	{
		double cost;

		cost = A_PRICE_MONTH;

		if (hours > A_CEILING)
		{
			cost = cost + (hours - A_CEILING) * A_PRICE_HOUR;
		}

		return cost;

	}

	/*************************************************
	 * calcB calculates the charges for package B
	 *
	 * @return The cost for package B
	 */
	public double calcB()
	{
		double cost;

		cost = B_PRICE_MONTH;

		if (hours > B_CEILING)
		{
			cost = cost + (hours - B_CEILING) * B_PRICE_HOUR;
		}

		return cost;

	}

	/*************************************************
	 * calcC calculates the charges for package C
	 *
	 * @return The cost for package C
	 */
	public double calcC()
	{
		return C_PRICE_MONTH;
	}

	/**
	 * calcTax calculates the tax on the passed charge
	 *
	 * @return The tax for this charge.
	 */
	public double calcTax()
	{
		return calcCost() * TAX_RATE;
	}

	/**
	 * saveWithB calculates whether or not this charge would be less if they
```

```
 * were on plan B
 *
 * @return true if you can save with B, false otherwise.
 */
public boolean saveWithB()
{
	boolean result;

	result = false;
	if (packageCode == 'A' || packageCode == 'a')
	{
		result = this.calcCost() > calcB();
	}
	return result;
}

/**
 * saveWithC calculates whether or not this charge would be less if they
 * were on plan C
 *
 * @return true if there are savings with C false otherwise
 */
public boolean saveWithC()
{
	boolean result;

	result = false;
	if (packageCode == 'A' || packageCode == 'B' || packageCode == 'a'
				|| packageCode == 'b')
	{
		result = this.calcCost() > calcC();
	}
	return result;
}

/**
 * savingsWithB calculates the savings with planB
 *
 * @return the amount of saving with B, 0 if no savings.
 */
public double savingsWithB()
{
	double result;

	result = 0.0;

	if (saveWithB())
	{
		result = this.calcCost() - this.calcB();
	}
	return result;
}

/**
 * savingsWithC calculates the savings if the charge would be less if they
 * were on plan C
 *
 * @return the amount of saving with C.
 */
public double savingsWithC()
{
	double result;

	result = 0.0;
```



```
	if (saveWithC())
	{
		result = this.calcCost() - this.calcC();
	}
	return result;
}

/************************************************
 * toString describes this charge. It should include the package for this
 * charge and the hours.
 *
 * @return a String representation of this package
 */
public String toString()
{
	return String.format("Package: %s\tHours: %f", this.packageCode,
		this.hours);
}

/**
 * needAddtlHours records whether or not additional hours are needed for
 * this package
 *
 * @return true if we need to include additional hours, false otherwise
 */
public boolean needAddtlHours()
{
	boolean addtl;

	if (packageCode == 'A' || packageCode == 'B')
		addtl = true;
	else
		addtl = false;
	return addtl;
}

/**
 * getAddtlHours calculates the additional hours based on this package code
 *
 * @return the additional hours
 */
public double getAddtlHours()
{
	double extra;
	extra = 0;

	if (needAddtlHours())
	{
		if (this.packageCode == 'A' && this.hours > this.A_CEILING)
		{
			extra = this.hours - this.A_CEILING;
		}
		else if (this.packageCode == 'B' && this.hours > this.B_CEILING)
		{
			extra = this.hours - this.B_CEILING;
		}
	}

	return extra;
}

/**
 * getAddtlCharge calculates the additional charge for this package
 *
```

```
 * @return this additional charge.
 */
public double getAddtlCharge()
{
	double extra;
	extra = 0;

	if (needAddtlHours())
	{
		if (this.packageCode == 'A' && this.hours > this.A_CEILING)
		{
			extra = getAddtlHours() * this.A_PRICE_HOUR;
		}
		else if (this.packageCode == 'B' && this.hours > this.B_CEILING)
		{
			extra = getAddtlHours() - this.B_PRICE_HOUR;
		}
	}

	return extra;
}

/**
 * getBase returns the base charge
 *
 * @return the base charge for this package
 */
public double getBase()
{
	double base;
	if (packageCode == 'C' || packageCode == 'c')
		base = this.C_PRICE_MONTH;
	else if (packageCode == 'B' || packageCode == 'b')
		base = this.B_PRICE_MONTH;
	else
		base = this.A_PRICE_MONTH;

	return base;
}

/**
 * getBaseHours returns the base hours for this package
 *
 * @return base hours.
 */
public double getBaseHours()
{
	double base;
	if (packageCode == 'A' || packageCode == 'a')
		base = this.A_CEILING;
	else
		base = this.B_CEILING;
	return base;
}

/**
 * getHours returns the total hours for this charge
 *
 * @return This charge's hours
 */
public double getHours()
{
	return hours;
}
```



```
	/**
	 * getPackage returns the standardized package code
	 *
	 * @return this package code
	 */
	public char getPackage()
	{
		return packageCode;
	}

	/**
	 * formatLabel prints a label right justified in a field width wide.
	 *
	 * @param label
	 *            The label that we want to print
	 * @param width
	 *            The width of the field
	 * @return text that is the label right justified in a field width wide.
	 */
	public static String formatLabel(String label, int width)
	{
		return String.format("%" + width + "s", label);
	}
}
```

```
/*********************************************************************
 * WordGuess is a class to supoort a Hangman  type game
 *
 * @author Nancy Harris, JMU
 * @version 04/15/2015
 * ****************************************/

public class WordGuess
{
	private final int NUM_STRIKES = 6;

	private String theWord;
	private String userWord;
	private String guesses;
	private int strikes;

	/*****************************************
	 * WordGuess constructor
	 *
	 * sets theWord to be the dictionary word and uses the makeUserWord method
	 * to set up the userWord for building It initializes the guesses and
	 * strikes to empty.
	 *
	 * @param dictWord
	 *            The dictionary word
	 ****************************************/
	public WordGuess(String dictWord)
	{
		theWord = dictWord;
		userWord = makeUserWord();
		guesses = "";
		strikes = 0;
	}

	/*****************************************
	 * getStrikes returns the current number of strikes
```

```
 * @return The current strikes
 ******************************************/

public int getStrikes()
{
	return strikes;
}

/******************************************
 * getTheWord returns the dictionary word
 *
 * @return The dictionary word
 ******************************************/
public String getTheWord()
{
	return theWord;
}

/******************************************
 * getUserGuesses returns a String that includes
 * the number of strikes and the current list of guesses
 * @return the strikes and guesses in a formatted String
 ******************************************/
public String getUserGuesses()
{
	return String.format("Strikes: %d\tGuesses: %s", strikes, guesses);
}

/******************************************
 * getUserWord returns the current state of the
 * user word
 * @return The user word
 ******************************************/
public String getUserWord()
{
	String builder;
	builder = "";

	for (int idx = 0; idx < userWord.length(); idx++)
	{
		if (idx == userWord.length() - 1)
			builder = builder + userWord.charAt(idx);
		else
			builder = builder + userWord.charAt(idx) + " ";
	}
	return builder;
}

/******************************************
 * isInWord returns true if the guess is in the
 * word and false otherwise.
 *
 * @param guess The current guess
 * @return True if the guess is in the word and
 * false otherwise.
 ******************************************/
public boolean isInWord(char guess)
{
	boolean result;
	result = false;

	if (theWord.contains(guess + ""))
		result = true;
```



```
	return result;
}

/******************************************
 * isOut returns true if the user is out of strikes
 * and false otherwise
 * @return true if the user is out of strikes and
 * false otherwise.
 ******************************************/
public boolean isOut()
{
	return strikes >= NUM_STRIKES;
}

/******************************************
 * isWordComplete determines if the user has
 * completely guessed the word.
 * @return True if the user has guessed the word,
 * false otherwise.
 ******************************************/
public boolean isWordComplete()
{
	return !userWord.contains("_");
}

/******************************************
 * makeUserWord makes the starting state of the
 * userWord (progress word) which is a String
 * that is theWord.length() long and filled with the
 * underscore (_) character.
 *
 * Method changed to private and void during
 * refactoring.
 ******************************************/
private void makeUserWord()
{
	String word;
	word = "";
	for (int idx = 0; idx < theWord.length(); idx++)
	{
		word = word + "_";
	}
}

/******************************************
 * updateUserWord updates the userWord (progress word)
 * with the new guess. This method will make no
 * change if the guess is not in the word.
 *
 * @param guess The guess that the user has
 * made
 * @return The new version of the userWord
 ******************************************/
public String updateUserWord(char guess)
{
	// System.out.println("In updateUW: " + userWord);
	String newWord;
	newWord = userWord;

	if (isInWord(guess))
	{
		newWord = "";
		for (int idx = 0; idx < theWord.length(); idx++)
		{
```

```
                        if (theWord.charAt(idx) == guess)
                        {
                                newWord = newWord + guess;
                        }
                        else
                        {
                                newWord = newWord + userWord.charAt(idx);
                        }
                }
        }
        userWord = newWord;

        return getUserWord();
    }

    /*****************************************
     * updateGuesses updates the list of guesses with
     * the current guess, and also determines if a strike
     * should be assessed.
     *
     * @param guess The guess that the user has made
     * @return The new version of getGuesses()
     *****************************************/
    public String updateGuesses(char guess)
    {
        int idx;

        // update strikes
        if (guesses.contains("" + guess))
                strikes++;
        else if (!isInWord(guess))
                strikes++;

        // update guesses
        if (guesses.length() == 0)
                guesses = guesses + guess;
        else
        {
                for (idx = 0; idx < guesses.length() && guesses.charAt(idx) < guess; idx++);

                if (idx == 0 && guesses.charAt(idx) != guess)
                        guesses = guess + guesses;
                else if (idx < guesses.length() && guesses.charAt(idx) != guess)
                        guesses = guesses.substring(0, idx) + guess
                                                + guesses.substring(idx);
                else if (idx == guesses.length())
                        guesses = guesses + guess;
                // else do nothing
        }
        return getUserGuesses();
    }

} // END class WordGuess
```