2. Consider a method of encoding and decoding words that is based on a *master string*. This master string will contain all the letters of the alphabet, some possibly more than once. An example of a master string is `"sixtyzipperswerequicklypickedfromthewovenjutebag"`. This string and its indexes are shown below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
s	i	x	t	y	z	i	p	p	e	r	s	w	e	r	e	q	u	i	c	k	l	y	p

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
i	c	k	e	d	f	r	o	m	t	h	e	w	o	v	e	n	j	u	t	e	b	a	g

An encoded string is defined by a list of *string parts*. A string part is defined by its starting index in the master string and its length. For example, the string `"overeager"` is encoded as the list of string parts [(37, 3), (14, 2), (46, 2), (9, 2)] denoting the substrings `"ove"`, `"re"`, `"ag"`, and `"er"`.

String parts will be represented by the `StringPart` class shown below.

```
public class StringPart
{
  /** @param start  the starting position of the substring in a master string
   *  @param length  the length of the substring in a master string
   */
  public StringPart(int start, int length)
  {  /* implementation not shown */  }


  /** @return  the starting position of the substring in a master string
   */
  public int getStart()
  {  /* implementation not shown */  }


  /** @return  the length of the substring in a master string
   */
  public int getLength()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

The class `StringCoder` provides methods to encode and decode words using a given master string. When encoding, there may be multiple matching string parts of the master string. The helper method `findPart` is provided to choose a string part within the master string that matches the beginning of a given string.

```
public class StringCoder
{
  private String masterString;

  /** @param master  the master string for the StringCoder
   *             Precondition: the master string contains all the letters of the alphabet
   */
  public StringCoder(String master)
  {  masterString = master;  }


  /** @param parts  an ArrayList of string parts that are valid in the master string
   *             Precondition: parts.size() > 0
   *  @return  the string obtained by concatenating the parts of the master string
   */
  public String decodeString(ArrayList<StringPart> parts)
  {  /* to be implemented in part (a) */  }


  /** @param str  the string to encode using the master string
   *             Precondition: all of the characters in str appear in the master string;
   *                           str.length() > 0
   *  @return  a string part in the master string that matches the beginning of str.
   *             The returned string part has length at least 1.
   */
  private StringPart findPart(String str)
  {  /* implementation not shown */  }


  /** @param word  the string to be encoded
   *             Precondition: all of the characters in word appear in the master string;
   *                           word.length() > 0
   *  @return  an ArrayList of string parts of the master string that can be combined
   *             to create word
   */
  public ArrayList<StringPart> encodeString(String word)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

GO ON TO THE NEXT PAGE.

AP® COMPUTER SCIENCE A
2008 SCORING GUIDELINES

Question 2: String Coder

Part A:	decodeString	**4 1/2 points**

- **+1** traverse `parts`
  - **+1/2** correctly access an element of `parts` (in context of loop)
  - **+1/2** access all elements of `parts` (lose this if index out-of-bounds)

- **+2** retrieve substrings from `masterString`
  - **+1/2** correctly call `getStart()` and `getLength()` on accessed part
  - **+1 1/2** extract a substring from `masterString`
    - **+1/2** `masterString.substring(X,Y)`
    - **+1** extract correct substring

- **+1 1/2** build and return decoded string
  - **+1** correctly build string from substrings of `masterString`
  - **+1/2** return built string

Part B:	encodeString	**4 1/2 points**

- **+1/2** construct an `ArrayList<StringPart>` (must assign to a variable, generic okay)

- **+3 1/2** find, collect string parts, and build list (in context of loop)
  - **+1** `findPart(X)`, where `X` is `word` or a substring of `word`
  - **+1** calls to `findPart` involve progressively smaller suffixes of `word`
  - **+1/2** add found string part to `ArrayList` of string parts
  - **+1** build correct list of string parts (must have used `findPart`)

- **+1/2** return `ArrayList` of string parts