Exam 2 Review
Objects, Arrays, Strings

Objects

- Declared using class statement.
- An object is created by invoking the class's constructor using the new keyword.
- An object is stored in a variable declared with the class as its type
- Values passed to a constructor must be copied to object fields to "stick" … why?

Declaring an Object Class

```
Tree myMaple;  // Variable defined as type Tree

void setup() {
  myMaple = new Tree("maple", 30.3);   // Create
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    ellipse(random(width),random(height),50,50);
  }
}
```

fields

constructor

method

Creating Objects (aka Object Instances)

1. Declare a variable with the class as type
2. Invoke the constructor using the new keyword and assign to variable

```
Tree myMaple;           // Variable defined as type Tree

myMaple = new Tree("maple", 30.3);   // Create and assign


// -----

// Two steps combined in one
Tree myMaple = new Tree("maple", 30.3);
```

Creating Objects

- What is wrong with this?

```
Tree myMaple;          // Variable defined as type Tree

void setup() {
  Tree myMaple = new Tree("maple", 30.3);   // Combined
}
```

Using Objects

- fields : variables 'owned by' an object
  - i.e. defined inside the class statement
- methods : functions 'owned by' an object

- A variable holding an object is used to scope access to the fields and methods of that particular object

Using Objects

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  myMaple.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Using Objects

What is wrong with this?

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  Tree.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Shapes, Inc.

We have been hired to model the business objects of Shapes, Inc. Following are their requirements.

1. All Shapes have an (x, y) position marking the Shape center
2. All Shapes are **red**
3. All Shapes respond to a request to display itself

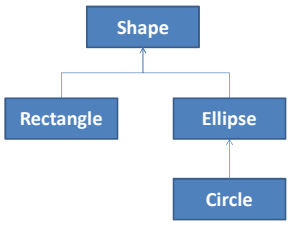
4. A Rectangle is a kind of Shape
5. An Ellipse is a kind of Shape
6. A Circle is a kind of Ellipse

7. An Ellipse turns white when the mouse hovers over it.
8. All Shapes can be dragged.

Questions

- What color is a Rectangle?
- How does a Circle specialize an Ellipse?
- What color is a Circle when the mouse is over it?

Modeling the Shapes, Inc. Business



A Shape Class

```
class Shape {
  float x;
  float y;
  color c;

  // Constructor
  Shape( float x, float y ) {
    this.x = x;
    this.y = y;
    this.c = color(255, 0, 0);
  }

 // Display the Shape
  void display() {
    fill(c);
    text("?", x, y);
  }
}
```

1. Shapes have a position
2. Shapes are red
3. Shapes respond to display

shapes1.pde

The `this` keyword

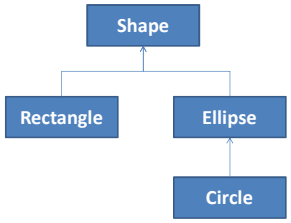
- Within an object, `this` is a shorthand for the object itself
- The most common use of `this` is to avoid a field access problems that occur due to shadowing
- The use of `this` explicitly changes the scope to the object level

- Reconsider the Shape constructor…

How to set up relationships?

Question:

If all Shapes have a position and all Shapes are red, how can we grant these properties to Rectangle and Ellipse, without reproducing them in every class?



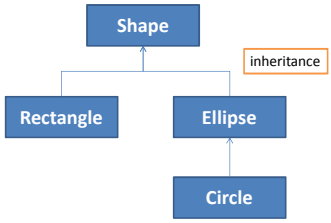
In a way, Rectangle and Ellipse `extend` the standard Shape object with specialized ways of displaying themselves.

How to set up relationships?

Answer:

We can set up an explicit relationship between Rectangle and Shape, and between Ellipse and Shape call ***Inheritance.***

This will automatically cause Shape fields and methods to be available to Rectangle and Ellipse.




```
class Rectangle extends Shape {
  float w;
  float h;

  Rectangle( float x, float y, float w, float h) {
    super(x, y);
    this.w = w;
    this.h = h;
  }

  // Display the Rectangle
  void display() {
    fill(c);
    rect(x, y, w, h);
  }
}
```

sets up the inheritance relationship

adds two new fields, width and height

invokes the superclass constructor

overrides the Shape display() method

Where does a Rectangle find x and y?

shapes2.pde

Inheritance – Some Terminology

- A new class (subclass) can be declared to extend the behavior of an existing class (superclass)
  - A subclass is aka: derived class, child class, …
  - A superclass is aka: base class, parent class,
- A subclass automatically gets access to (i.e. inherits) all members of the superclass
  - Members include both fields and methods
- A subclass can override the members of its superclass by re-declaring them
  - Think of variable shadowing, but now for methods too

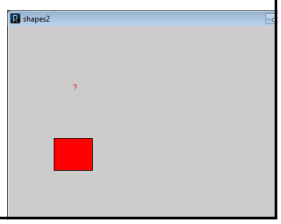
The `super` keyword

- Within an object, `super` is a shorthand for the superclass of the current object
- The most common use of `super` is to invoke a superclass constructor
- The use of `super` explicitly changes the scope to the superclass level

Test it

```
void setup() {
  size(500, 500);

  Shape     s = new Shape(100, 100);
  Rectangle r = new Rectangle (100, 200, 60, 50);

  s.display();
  r.display();
}
```

Note: The Rectangle knows where to draw itself, even though it does not have an x or y field. It inherits x and y from Shape.



shapes2.pde

```
class Shape {
  float x;
  float y;
  color c;

  // Constructor
  Shape( float x, float y ) {
    this.x = x;
    this.y = y;
    this.c = color(255, 0, 0);
  }

 // Display the Shape
  void display() {
    fill(c);
    text("?", x, y);
  }
}
```

```
class Rectangle extends Shape {
  float w;
  float h;

  // Constructor
  Rectangle( float x, float y,
             float w, float h) {
    super(x, y);
    this.w = w;
    this.h = h;
  }

  // Display the Rectangle
  void display() {
    fill(c);
    rect(x, y, w, h);
  }
}
```

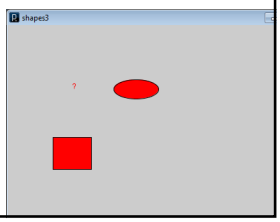
The Ellipse Class

```
class Ellipse extends Shape
{
  float w;
  float h;

  Ellipse( float x, float y, float w, float h) {
    super(x, y);
    this.w = w;
    this.h = h;
  }

  // Display the Ellipse
  void display() {
    fill(c);
    ellipse(x, y, w, h);
  }
}
```

shapes3.pde

Test it

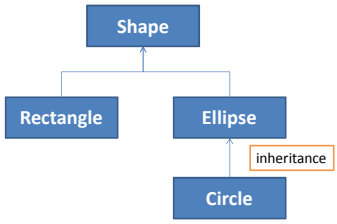
```
void setup() {
  size(500, 500);
  smooth();
  ellipseMode(CENTER);
  rectMode(CENTER);

  Shape     s = new Shape(100, 100);
  Rectangle r = new Rectangle (100, 200, 60, 50);
  Ellipse   e = new Ellipse(200, 100, 70, 30);

  s.display();
  r.display();
  e.display();
}
```



shapes3.pde

Inheritance, Cont'd

- Inheritance hierarchies can be used to establish multiple layers of objects



The Circle Class

```
class Circle extends Ellipse
{
  float r;

  Circle( float x, float y, float r ) {
    super(x, y, 2*r, 2*r);
    this.r = r;
  }
}
```

adds only a radius field

translates radius to Ellipse constructor width and height arguments

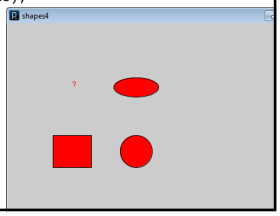
shapes4.pde

```
class Ellipse extends Shape
{
  float w;
  float h;

  // Constructor
  Ellipse( float x, float y,
           float w, float h) {
    super(x, y);
    this.w = w;
    this.h = h;
  }

  // Display the Ellipse
  void display() {
    fill(c);
    ellipse(x, y, w, h);
  }
}
```

```
class Circle extends Ellipse
{
  float r;

  // Constructor
  Circle( float x, float y,
          float r ) {
    super(x, y, 2*r, 2*r);
    this.r = r;
  }

  // Does not override display

}
```

Test it

```
void setup() {
  size(500, 500);
  smooth();
  ellipseMode(CENTER);
  rectMode(CENTER);

  Shape     s = new Shape(100, 100);
  Rectangle r = new Rectangle(100, 200, 60, 50);
  Ellipse   e = new Ellipse(200, 100, 70, 30);
  Circle    c = new Circle(200, 200, 25);

  s.display();
  r.display();
  e.display();
  c.display();
}
```



shapes4.pde

Polymorphism

poly = many, *morph* = form

In Biology, when there is more than one form in a single population


Light-morph jaguar (typical)


Dark-morph or melanistic jaguar (about 6% of the South American population)

In Computing, we have two common types of Polymorphism

1. Signature Polymorphism
2. Subtype Polymorphism

http://en.wikipedia.org/wiki/Polymorphism_%28biology%29

Signature Polymorphism

- It is possible to define multiple functions with the same name, but different signatures.
  - A *function signature* is defined as
    - The function name, and
    - The order and type of its parameters
- Consider the built-in color() function …

```
color(gray)
color(gray, alpha)
color(value1, value2, value3)
color(value1, value2, value3, alpha)
…
```

Signature Polymorphism

```
void draw() { }

void mousePressed() {
  int i;
  i = 10;
  i = increment(i, 2);
  //i = increment(i);
  println(i);
}

// increment a variable
int increment(int j, int delta) {
  j = j + delta;
  return j;
}

int increment(int k) {
  k = increment(k, 1);
  return k;
}
```

In this case it is said that the increment function is ***overloaded***

Subtype Polymorphism

- Inheritance implements Subtype Polymorphism
  - A Rectangle is a type of Shape
  - An Ellipse is a type of Shape
  - A Circle is a type of Ellipse
- Implication:
  - A Rectangle can be stored in a variable of type Shape
  - What about Ellipses, Circles?

Using Subtype Polymorphism

Store everything that is a type of Shape in an array of Shapes.

```
Shape[] shapes = new Shape[3];

void setup() {
  size(500, 500);
  smooth();
  ellipseMode(CENTER);
  rectMode(CENTER);

  shapes[0] = new Rectangle(100, 200, 60, 50);
  shapes[1] = new Ellipse(200, 100, 70, 30);
  shapes[2] = new Circle(200, 200, 25);

  for (int i=0; i<shapes.length; i++) {
    shapes[i].display();
  }
}
```

an array of Shapes

all objects that are Shape subclasses can be stored in the array, even Circle

now we can use a loop

shapes5.pde

containsPoint()

- Let's give each shape a containsPoint() method that returns a boolean
  - Returns true if the shape contains a given point
  - Returns false otherwise
- Each subclass must implement a different version of containsPoint() because each uses a different calculation.

containsPoint() for Shape

- By default, the abstract Shape object cannot determine if it contains a point
- Always return false

```
class Shape {
  …
  // Test if a point is within a Shape
  boolean containsPoint( float x, float y ) {
    return false;
  }
}
```

shapes6.pde

containsPoint() for Rectangle

- Test the location of the point wrt the locations of Rectangle sides

```
class Rectangle extends Shape {
  …
  // containsPoint() for Rectangle
  boolean containsPoint( float x, float y ) {
    float w2 = 0.5*w;
    float h2 = 0.5*h;
    if (x < this.x-w2) { return false; }
    if (x > this.x+w2) { return false; }
    if (y < this.y-h2) { return false; }
    if (y > this.y+h2) { return false; }
    return true;
  }
}
```

shapes6.pde

containsPoint() for Ellipse

- Use a special formula to determine if a point is in an Ellipse

```
class Ellipse extends Shape {
  …
  // containsPoint() for an Ellipse
  boolean containsPoint( float x, float y ) {
    float dx = x - this.x;
    float dy = y - this.y;
    float hw = 0.5*w;
    float hh = 0.5*h;
    if ( (dx*dx)/(hw*hw) + (dy*dy)/(hh*hh) < 1.0 ) {
      return true;
    } else {
      return false;
    }
  }
}
```

shapes6.pde

containsPoint() for Circle

- Test the distance between the point and the Circle center to see if it is less than the radius

```
class Circle extends Ellipse {
  …
  // containsPoint() for a Circle
  boolean containsPoint( float x, float y ) {
    if ( dist(this.x, this.y, x, y) < r ) {
      return true;
    } else {
      return false;
    }
  }
}
```

shapes6.pde

All Subclasses Get Superclass Methods

- Add a method to Shape that changes the fill color to white when the mouse is over the Shape
- Use containsPoint() to test this condition
- Plan
  1. Move the display() loop from setup() to draw()
  2. Add a mouseMoved() method to Shape that changes fill color based on containsPoint()
  3. Call all Shape class mouseMoved() methods from top-level mouseMoved().

New Top-level Program

```
Shape[] shapes = new Shape[3];

void setup() {
  size(500, 500);
  smooth();
  ellipseMode(CENTER);
  rectMode(CENTER);

  shapes[0] = new Rectangle (100, 200, 60, 50);
  shapes[1] = new Ellipse(200, 100, 70, 30);
  shapes[2] = new Circle(200, 200, 25);
}

void draw() {
  background(200);
  for (int i=0; i<shapes.length; i++) {
    shapes[i].display();
  }
}

void mouseMoved() {
  for (int i=0; i<shapes.length; i++) {
    shapes[i].mouseMoved();
  }
}
```

display loop moved to draw()

mouseMoved() called for all Shapes

shapes6.pde

mouseMoved() method for Shape

- Uses containsPoint() to decide how to change fill color
- Note: The appropriate subclass implementation of containsPoint() will be invoked, depending upon the type of Shape subclass on which the method is invoke upon

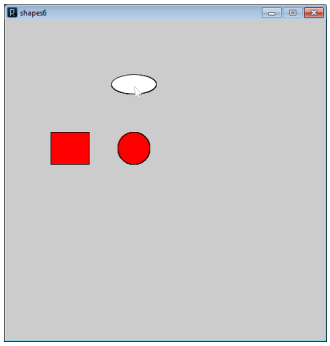
This is declared in the Shape class, but …

```
class Shape {
  …
  void mouseMoved() {
    if ( containsPoint( mouseX, mouseY ) == true ) {
      this.c = color(255);
    } else {
      this.c = color(255, 0, 0);
    }
  }
}
```

… this is invoked on the subclass that overrides it.

shapes6.pde

Test it



shapes6.pde

- But wait, only Ellipse objects are supposed to turn white on mouse over, not Rectangles
- Overriding a method can also be used to cancel default behavior.
- Add the following method to Rectangle to override the Shape class mouseMoved() to replace behavior

```
void mouseMoved() {
   // Do nothing
}
```

shapes6.pde

Arrays - Creating

- A structure that can hold multiple items of a common data type
- Arrays can hold any data type, including objects
- The data type to be held by an array must be declared as part of the array declaration
- Arrays are themselves a kind of type, which is made by adding brackets to the type that the array can hold

Arrays – Creating and Init'ng (3 Steps)

1. Declare an array variable
   - The variable is NOT an array
2. Create an array and assign it to the variable
   - Use the new keyword and size
   - The array is filled with default values
     - int <- 0
     - float <- 0.0
     - boolean <- false;
     - any object including String <- null
3. Fill the array with items of appropriate type

```
Tree[] trees;
```

Step 1

trees ← No array. Only a variable that can hold an array.

```
Tree[] trees;
trees = new Tree[5];
```

Step 2

	trees
0	null
1	null
2	null
3	null
4	null

← An empty array. null Tree objects.

```
Tree[] trees;
trees = new Tree[5];
trees[0] = new Tree("maple", 20.0);
trees[1] = new Tree("oak", 203.4);
```

Step 3

	trees
0	name="maple"; height=20.0;
1	name="oak"; height=203.4;
2	null
3	null
4	null

← An array with two Tree objects.

```
Tree[] trees;
trees = new Tree[5];
for (int i=0; i<5; i++) {
  trees[i] = new Tree( "maple"+i, random(200.0) );
}
```

Step 3

	trees
0	name="maple0"; height=12.5;
1	name="maple1"; height=105.3;
2	name="maple2"; height=198.6;
3	name="maple3"; height=4.08;
4	name="maple4"; height=99.9;

← An array with five Tree objects.

```
int[] ages;
```

Step 1

ages ← No array. Only a variable that can hold an array.

```
int[] ages;
ages = new int[5];
```

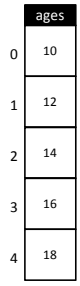
Step 2

	ages
0	0
1	0
2	0
3	0
4	0

← An empty array. Default ints (0).

```
int[] ages;
ages = new int[5];
for (int i=0; i<5; i++) {
  ages[i] = 10 + 2*i;
}
```

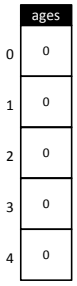
Step 3



← An array with five integers.

```
int[] ages = new int[5];

// Same as
// int[] ages;
// ages = new int[5];
```

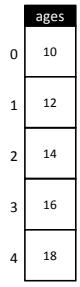
Step 1+2



← An empty array. Default ints (0).

```
int[] ages = new int[] {10, 12, 14, 16, 18};

// Same as
// int[] ages = new int[5];
// for (int i=0; i<5; i++) { ages[i] = 10 + 2*i; }
```

Step 1+2+3



← An array with five integers.

Arrays – Using

- An item in an array is accessed by following an array variable with square brackets containing the item number (index)
- The result of the array accessor expression is the item in the array at the index
- Array indexes start with 0
- Once accessed with brackets, the result can be used as if it was the item at the location in the array

```
Tree[] trees;

void setup() {
  trees = new Tree[3];
  trees[0] = new Tree("maple", 30.3);
  trees[1] = new Tree("oak", 130.3);
  trees[2] = new Tree("spruce", 230.3);
}

void draw() {
  for (int i=0; i<trees.length; i++ ) {
    trees[i].draw();
  }
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    ellipse( random(width), random(height), 50, 50 );
  }
}
```

Arrays of arrays (2D Arrays)

- If an array can be made of any type by adding brackets, and …
- an array is a kind of type, then …
- an array of arrays should be possible by adding a second set of brackets

```
boolean[] cell1;     // A variable that holds an array of booleans

boolean[][] cell2;   // A variable that holds an array of
                     // boolean arrays
```

```
boolean[] cell1;
cell1 = new boolean[5];
```

cell1

0	false
1	false
2	false
3	false
4	false

← One-dimensional array

```
boolean[][] cell2;
cell2 = new boolean[4][5];
```

cell2

	0	1	2	3	4
0	false	false	false	false	false
1	false	false	false	false	false
2	false	false	false	false	false
3	false	false	false	false	false

← Two-dimensional array

… an array of arrays

```
boolean[][] cell2;
cell2 = new boolean[4][5];

cell2[1][2] = true;
```

cell2

	0	1	2	3	4
0	false	false	false	false	false
1	false	false	true	false	false
2	false	false	false	false	false
3	false	false	false	false	false

Accessing all elements of the cell2 2D array

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[4][5];

  cell2[1][2] = true;

  for (int i=0; i < cell2.length; i++) {
    for (int j=0; j < cell2[i].length; j++) {
      println( cell2[i][j] );
    }
  }
}
```

What is going on here?

"Ragged" Arrays

```
float[][] ragged;

void setup() {

  ragged = new float[5][];

  for (int i=0; i<5; i++) {
    int n = int(random(10));
    ragged[i] = new float[n];
  }

  for (int i=0; i<5; i++) {
    println(ragged[i].length);
  }

}
```

ragged

	0	1	2	3	4
0	1.23	3.25	9.84		
1	8.87	6.70	5.10	0.59	4.44
2	9.01	4.98			
3	8.50				
4	4.79	8.11	0.98	1.87	

ex3.pde

Proving a 2D array is an array of arrays

- Access fields and methods of top-level array

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][5];   // Create array of arrays

  println( cell2[0].length );  // Access array

  cell2[1][2] = true;          // Access array in array
  println( cell2[1] );         // Access array
}
```

```
5
[0] false
[1] false
[2] true
[3] false
[4] false
```

Proving a 2D array is an array of arrays

- Build a "ragged array"

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][];

  cell2[0] = new boolean[2];
  cell2[1] = new boolean[4];
  cell2[2] = new boolean[1];

  println("---");
  println(cell2[0]);
  println("---");
  println(cell2[1]);
  println("---");
  println(cell2[2]);
  println("---");
  println(cell2[3]);
  println("---");
  println(cell2[4]);
}
```

```
---
[0] false
[1] false
---
[0] false
[1] false
[2] false
[3] false
---
[0] false
---
null
---
null
```

ArrayList

- Constructors

```
ArrayList myList = new ArrayList();
ArrayList myList = new ArrayList(initialSize);
```

- Fields
- Methods

```
myList.size()                 // Returns the num of items held.
myList.add(Object o)          // Appends o to end.
myList.add(int idx, Object o) // Inserts o at pos idx.
myList.remove(int idx)        // Removes item at pos idx.
myList.get(int idx)           // Gets items at idx. No removal.
myList.set(int idx, Object o) // Replaces item at idx with o.
myList.clear()                // Removes all items.
myList.isEmpty()              // Returns true if empty.
```

HashMap

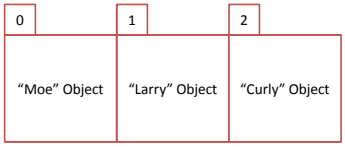
- Constructors

```
HashMap myMap = new HashMap();
HashMap myMap = new HashMap(initialCapacity);
```

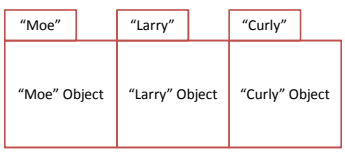
- Fields
- Methods

```
myMap.size()                    // Returns num of items held.
myMap.put(Object key, Object o) // Puts o in map at key
myMap.remove(Object key)        // Remove Object at key
myMap.get(Object key)           // Get Object at key
myMap.containsKey(Object key)   // True if map contains key
myMap.containsValue(Object val) // True if map contains val
myMap.clear()                   // Removes all items.
myMap.isEmpty()                 // Returns true if empty.
```

```
ArrayList stooges = new ArrayList();

void setup() {
  stooges.add( new Stooge("Moe" ));
  stooges.add( new Stooge("Larry" ));
  stooges.add( new Stooge("Curly" ));

  println( stooges.size() );
  Stooge s = (Stooge)stooges.get(0);
  println(s.name);
}

class Stooge {
  String name;

  Stooge( String name ) {
    this.name = name;
  }
}
```




```
HashMap stooges = new HashMap();

void setup() {
  stooges.put( "Moe",   new Stooge("Moe" ));
  stooges.put( "Larrry", new Stooge("Larry" ));
  stooges.put( "Curly",  new Stooge("Curly" ));

  println( stooges.size() );
  Stooge s = (Stooge)stooges.get("Moe");
  println(s.name);
}

class Stooge {
  String name;

  Stooge( String name ) {
    this.name = name;
  }
}
```



Strings

- Declaring String objects with no chars

```
String myName;
String myName = new String();
```

- Declaring String objects init'd w/ char array

```
String myName = "Fred";
String myName = new String("Fred");
```

String class methods

- `charAt(index)`
  - Returns the character at the specified index
- `equals(anotherString)`
  - Compares a string to a specified object
- `equalsIgnoreCase(anotherString)`
  - S/A ignoring case (i.e. 'A' == 'a')
- `indexOf(char)`
  - Returns the index value of the first occurrence of a character within the input string
- `length()`
  - Returns the number of characters in the input string
- `substring(startIndex, endIndex)`
  - Returns a new string that is part of the input string
- `toLowerCase()`
  - Converts all the characters to lower case
- `toUpperCase()`
  - Converts all the characters to upper case
- `concat(anotherString)`
  - Concatenates String with anotherString

http://docs.oracle.com/javase/7/docs/api/

Try it!

```
String s1 = "abcdefg";
println( s1.charAt(0) );
```

```
String s1 = "abcdefg";
String s2 = "abcdefg";
if (s1.equals(s2)) println("They are equal");
```

```
String s1 = "abcdefg";
println( s1.indexOf('c') );
```

```
String s1 = "abcdefg";
println( s1.substring(2, 5) );
```

```
println( "abcdefg".length() );
```

```
println( "abcdefg".toUpperCase() );
```

Building Strings – Use '+'

```
void setup() {
  String s1 = "Hello";
  String s2 = "World";
  String s3 = s1 + " " + s2;
  println( s3 );
}
```

```
void setup() {
  String s1 = "She is number ";
  String s2 = " in computer science.";
  String s3 = s1 + 1 + s2;
  println( s3 );
}
```

Numbers are converted to Strings prior to concatenation

Strings can be held by Arrays

- (Just like any other object or primitive type)

```
String[] tokens = new String[5];

void setup() {

  tokens[0] = "one";
  tokens[1] = "two";
  tokens[2] = "three";
  tokens[3] = "four";
  tokens[4] = "five";

  println(tokens);
}
```

```
[0] "one"
[1] "two"
[2] "three"
[3] "four"
[4] "five"
```

Strings can be held by Arrays

- Initialized when declared

```
String[] tokens = new String[] {"one", "two", "three", "four", "five"};

void setup() {
  println(tokens);
}
```

```
[0] "one"
[1] "two"
[2] "three"
[3] "four"
[4] "five"
```

Strings can be held by Arrays

- Not initialized

```
String[] tokens = new String[5];

void setup() {
  println(tokens);
}
```

```
[0] null
[1] null
[2] null
[3] null
[4] null
```

Built-in String functions (not methods)

split(*bigString, splitChar*)
- Breaks a String into a String Array, splitting on splitChar
- Returns new String Array

splitTokens(*bigString, splitCharString*)
- Breaks a String into a String Array, splitting on any char in splitCharString

join(*stringArray, joinChar*)
- Builds a new String by concatenating all Strings in stringArray, placing joinChar between each
- Inverse of split() function

text(*theString, x, y*)

text(*theString, x, y, width, height*)
- Draws *theString* on the sketch at (x, y)

Split a String based on a single or multiple separator chars

```
String s1 = "12, 34, 56";
String[] as;

void setup() {
  as = split(s1, ",");
  println( as );
}
```

```
[0] "12"
[1] " 34"
[2] " 56"
```

Creates String array
… no "new" statement

```
String s1 = "Data: 12, 34, 56";
String[] as;

void setup() {
  as = splitTokens(s1, ":,");
  println( as );
}
```

```
[0] "Data"
[1] " 12"
[2] " 34"
[3] " 56"
```

Join a String Array with a join char

```
String[] as = new String[] {"one", "two", "buckle my shoe"};

void setup() {
  String s1 = join( as, " | " );
  println( s1 );
}
```

```
one | two | buckle my shoe
```

Given the commands:

```
String aPalindrome = "a man, a plan, a canal Panama";
String[] strs = splitTokens(aPalindrome, ",");
```

Answer the following questions:

(3 pts) What will be the length of strs?

a) 1
b) 2
c) 3
d) 4

(3 pts) What will be the value of strs[1]?

a) "a man"
b) "a plan"
c) "a canal Panama"
d) 3

(3 pts) Write the expression used to obtain the number of elements in strs.