AP CS, Garfield HS – Earl Bergquist

Classes & Object: Vocabulary & Concepts Summary*

Class: A program entity that represents either:

1. A program / module, or
2. A template for a new type of objects.

A**bstraction**: A distancing between ideas and details. We can use Objects without knowing how they work, just like you don't need to know how an iPod's circuitry works just how to use it's control panel (methods) and the settings (fields).

Client Program (Class): A program that uses Objects
Object (Class): An entity that combines state and behavior.

1. **State**: Fields (i.e. int, double, array, other objects)
2. **Behavior**: Methods
   a. **mutator**: A method that modifies an object's state. Examples: setLocation, translate. Typically has a void return type
   b. **accessor**: A method that lets clients examine object state. Examples: distance, distanceFromOrigin. Typically has has a non-void return type

Object States: fields: A variable inside an object that is part of its state.

```
public class ObjectName {
          type name1;     // field1
          type name1;     // field2
      }
```

Creating an instance of an Object:

```
ObjectName objectInstance = new ObjectName();   // More on optinal parameters
later
```

For example for Object `Point`:

```
Point p1 = new Point();
```

Accessing an Objects Fields:

access: `objectInstance.field`
modify: `objectInstance.field = value;`

For example for Object `Point`:

access: `p1.x`
modify: `p1.y = 6;`

Object Behavior: Methods:
Instance method (or **Object method**): Exists inside each object of a class and gives behavior to each object. NOTE: no "static" in declaration.

```
public type name(parameters) {
    statements;
}
```

(continued next page)

**Assembled from Supplement Lesson slides of: Building Java Programs, Chapter 8 by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)*

AP CS, Garfield HS – Earl Bergquist

Kinds of Object Methods:

- **mutator**: A method that modifies an object's state.
  - Examples: setLocation, translate
- **accessor**: A method that lets clients examine object state.
  - Examples: distance, distanceFromOrigin
  - often has a non-void return type

Implicit parameter: The object on which an instance method is called.

```
p1.draw(g);
```

the object referred to by p1 is the implicit parameter, the method will act on its (p1's) fields

The instance method can refer to it's Object's fields

Constructor (method): Initializes the state of new objects. (where type is the Object's name)

```
public type(parameters) {
    statements;
}
```

A class can have **multiple constructors**. But each one must accept a unique set of parameters. BTW, this is also true of methods in Object Classes and Client Programs.

Array of Objects: you can create an array of any kind of objects, but the elements of an array of objects are initialized to null.
null : A value that does not refer to any object (yet).

dereference: To access field data or methods of an object with the dot notation:
such as a field: p1.x or method: s.length()
–It is illegal to dereference null (causes an exception).
–null is not any object, so it has no methods or data.

Creating an Array of Objects requires two-phase initialization:
1) initialize the array itself (each element is initially null)
2) initialize each element of the array to be a new object

String Example:

```
String[] words = new String[4];            // phase 1
for (int i = 0; i < words.length; i++) {
    coords[i] = "word" + i;                // phase 2
}
```

Point Object Example:

```
int numCities= scan.nextInt();
Point[] cities = new Point[numCities];      // phase 1
for (int i = 0; i < cities.length; i++) {
    int xvalue = scan.nextInt();
    int yvalue = scan.nextInt();
    Point[i] = new Point(xvalue, yvalue); // phase 2
}
```

**Assembled from Supplement Lesson slides of: Building Java Programs, Chapter 8 by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)*

Arrays as Object Fields: when the length of an Object's array field is not known till constructed:

• First just declare it like you would any variable with no size:

```
// declare our fields (data)
     private int numClasses;
     private int [] scores; // Note: No Size
```

• And then "Instantiate" it in the constructor method using the length available parameter.

```
  // Constructor for Object GradeBook
     public GradeBook (int numClasses) {
        this.numClasses = numClasses;
// We now "Instantiate" the array to its size
        scores = new int [numClasses];
     }
```

`toString` method - tells Java how to convert an object into a String

```
     public String toString() {
         code that returns a String representing this object;
     }
```

Every class has a toString, even if it isn't in your code. Default is: class's name @ object's memory address (base 16) for example: `Point@9e8c34`

Encapsulation: Hiding implementation details from clients using Private Fields.

- Encapsulation forces *abstraction*.
- separates external view (behavior) from internal view (state)
- protects the integrity of an object's data

Private Field: A field that cannot be accessed from outside the class

private type name; (see examples below to create methods to set and get private variables)

Shadowing: 2 variables with same name in same scope. Normally illegal, except when one variable is a field.

```
public class Point {
        private int x;
        private int y;

        public void setLocation(int x, int y) {
            ...
        }

        public int getX() {
            return x;
        }

        public int getY() {
            return y;
        }
}
```

**Assembled from Supplement Lesson slides of: Building Java Programs, Chapter 8 by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)*

this *:* Refers to the implicit parameter inside your class. (a variable that stores the object on which a method is called) Can:

- Refer to a field: this.**field**
- Call a method: this.**method**(**parameters**);
- One constructor can call another: this(**parameters**);

"this" can *fix* (clarify) variable shadowing:

```
public void setLocation(int x, int y) {
    this.x = x;
    this.y = y;
}
```

Here's a clever use of this in constructors:

```
public Point() {
    this(0, 0);     // calls (x, y) constructor
}

public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

- Avoids redundancy between constructors
- Only a constructor (not a method) can call another constructor