Card Bingo Questions

Would using `ArrayList` simplify this program?

- `ArrayList` is simpler for collections where insert, remove operations are needed
- arrays are simpler for fixed-sized collections and random access

How do you shuffle arrays?

- not needed here – we only need to shuffle decks
- we'll see later on – library method, algorithm

Objects and Classes

Objects and classes are the next step in organizing programs and building modules –

- we can group subroutines and variables that together have a single whole purpose into an *object*
  - an object is a black box which contains some state (values), with certain ways to access or manipulate that state
- objects in a program are used to represent real-world objects (and non-tangible things)
  - the object's state represents the real object's properties
  - the object's operations manipulate its state in the way that you interact with the real world object and manipulate its properties
- a *class* defines an object's properties and operations
  - a class provides a definition for a user-defined type
    - a *type* involves a set of legal values and the operations that can be applied to those value
  - an object is a particular *instance* of a class

Writing Classes

In Java, a class generally has one of two purposes –

- a holder of subroutines (such as `main`)
  - all elements (subroutines, global variables, global constants) are `static`
- a blueprint for creating objects
  - most elements are not `static` (exception is global constants)

Writing Classes

Elements of a class used to define objects –

- instance variables
  - these define the object's state – values that can be different for different objects and/or different at different times for one object
- one or more constructors
  - to initialize the instance variables
- methods
  - these define the operations that can be used to access and manipulate the object's state
  - may include getters and setters

Writing Classes – Syntax

- each public class goes in its own file, with the class name matching the file name

```
/**
 * Describe the purpose of the class.  (What
 * kind of thing does this class describe?)
 *
 * @author author's name
 */
public class ClassName {
  …
}
```

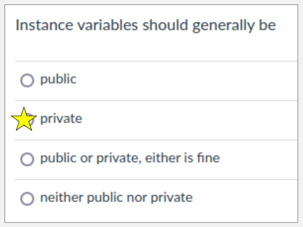
  - convention is to start class names with a capital letter (to distinguish from primitive types)

Writing Classes – Syntax

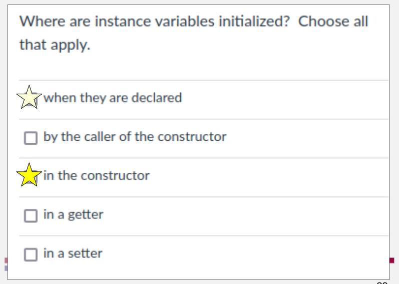
- instance variables define the object's state

```
public class ClassName {
  private type varname_; // description
  …
}
```

  - typically `private` rather than `public` (for encapsulation and information hiding)
  - not `static`
  - naming conventions
    - start with lowercase letter
    - end with _ to distinguish from local variables and parameters (note: this convention is not used in the book)
  - in some cases can be initialized at the point of declaration but more typically initialized in the constructor


Instance variables should generally be

- public
- [✓ private]
- public or private, either is fine
- neither public nor private


Where are instance variables initialized? Choose all that apply.

- [✓ when they are declared]
- by the caller of the constructor
- [✓ in the constructor]
- in a getter
- in a setter

Writing Classes – Syntax

- constructors create new objects
  - responsible for any setup that is required before an object can be used – typically initializing instance variables

```
public class ClassName {
  /**
   * Description.
   */
  public ClassName ( param-list ) {
    …
  }
}
```

also include @param tags for each parameter

  - not `static`
  - no return type or value (not even `void`)
  - constructor name must match the class name
  - can have any number of parameters (*default constructor* if 0)
  - can have multiple constructors but it must be possible to distinguish them by the number and/or type of their parameters

Writing Classes – Syntax

- methods implement operations
  - access and/or manipulate object's state

```
public class ClassName {
  /**
   * Description.
   */
  public return-type name ( param-list ) {
    …
  }
}
```

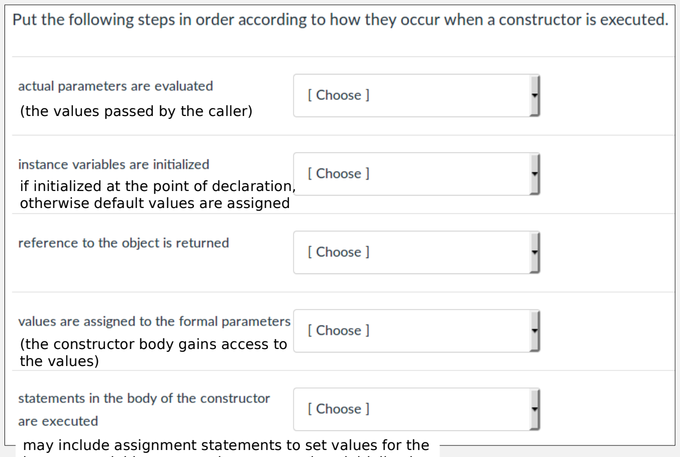
  - `public` methods are intended for use outside the class
  - `private` helper methods support the implementation of other methods but are not available outside the class
  - not `static`
  - naming conventions – generally same as subroutines/functions
    - getters – `get`*`Something`* (`is`*`Something`* for `boolean` return values)
    - setters – `set`*`Something`*

Initialization of Instance Variables

- there are three places an instance variable can be assigned a value
  - in the declaration
  - in the constructor
  - in a setter or other method
- guidelines
  - all variables must be initialized before they can be used, and only the constructor is guaranteed to be called before another method
  - to avoid sequencing problems, instance variables should generally be initialized in the declaration or the constructor
    - in the constructor is always possible
    - must be in the constructor if the value can't be hardcoded
    - for consistency, always initialize in the constructor

```
public class Demo {
  private int a_ = 10;
  private int b_;

  public Demo ( int value ) {
    b_ = value;
  }

  public void setA ( int value ){
    a_ = value;
  }

  public void setB ( int value ){
    b_ = value;
  }

  public void increment () {
    a_++;
    b_++;
  }
}
```

Constructors – Semantics



Put the following steps in order according to how they occur when a constructor is executed.

2	actual parameters are evaluated (the values passed by the caller)	[Choose]
1	instance variables are initialized if initialized at the point of declaration, otherwise default values are assigned	[Choose]
5	reference to the object is returned	[Choose]
3	values are assigned to the formal parameters (the constructor body gains access to the values)	[Choose]
4	statements in the body of the constructor are executed	[Choose]

may include assignment statements to set values for the instance variables – overwrites any previous initialization

`static`

A good rule of thumb –

- for classes used as a holder of subroutines (such as main), all elements are `static`
- for classes used as blueprints for objects, only global constants are `static`

The meaning of `static` –

- `static` means there is only one copy for the program (shared by all objects of that type)
- non-`static` means that each object has its own copy