* Arrays as Instance Variables
* Working with Strings
* Inheritance, Method Overriding, and Polymorphism
* Abstraction (Classes and Methods)
* Interfaces
* Association, Aggregation, and Composition
* Exception Handling
* Files
* You can instantiate a String object from a string value or from an array of characters
* To create a String from a string literal, use the following
  + String aString = new String(“This is a String”);
* To create a String from an array of characters, use the following
  + Char data[] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’};
  + String aString = new String(data);
* Methods of the String class are primarily instance methods – therefore called “attached” to an object
* Methods of the Character class are static therefore called “attached” to the class name
* Assume you forgot to initialize the array in the constructor
  + What will happen?
    - Run time error!
      * When you run the program and enter in the first grade, a runtime error will occur(NullPointerException)
      * No space in memory has been initialized to store the array contents!
* Remember, when passing arrays, you are not passing the contents of the array, only a reference value!
* All constructors that accept an array as a parameter for an array as an instance variable should use a copy of the array and not a reference value to the array
* All accessor methods that access an array as an instance variable should return a copy of the array and not a reference value to the array
* Mutators for Arrays as Instance Variables
  + Similar to accessors, all mutator methods that accept an array as a parameter for an array as an instance variable should use a copy of the array to store values and not the original array
* Inheritance and Composition
* “is-a” (Inheritance of “has-a”(Composition) can be used to determine the relationship between two classes
  + CheckingAccount is-a BankAccount
  + Apple is-a fruit
  + Car is-a Vehicle
  + Kitchen has-an oven
  + Ferrari has-an engine
* Provides an image of the logical design to help formulate the object-oriented design
* Provides easier maintenance and less duplication than maintaining individual classes
* Inheritance
* 
* Information hiding
  + Keeping data private
  + Data can only be altered by methods you choose and only in ways that you control
  + With inheritance, private variables of the superclass are not inherited
* Instance Variables
  + Avoid using protected instance variables in a superclass
  + Use non-private methods that access private instance variables
    - Provide public get and set methods
    - Helps to allow changes to the superclass use of these instance variables without affecting subclass implementations
    - Help to ensure that objects of the superclass maintain consistent states
* Constructors and Inheritance
  + A subclass does not inherit the constructor of a superclass
    - Constructors are not members of a class
      * Only members of a class are inheritable
    - Logically may cause issues in the program design
      * The parameters needed to create an object of the subclass may be different than those to create an object of the superclass
* Subclass Constructor Creation
  + Create a constructor using the process already learned
  + Using the constructor from the superclass by creating a constructor in the subclass and using the super keyword
* Overloading
  + Two methods share the same name, but have different formal parameters in the method header
* Overriding
  + A subclass defines a method with the same name and exactly the same parameters as the superclass method
* A superclass reference value may reference a subclass object
  + Why? Polymorphism
* A subclass reference value may not reference a superclass object
* Abstract Classes
  + A class that can be instantiated is called a concrete class
  + Sometimes it is desirable to force a class to be extended
  + Within an abstract class, it is also sometimes desirable to force a specific method to be overridden
  + To declare a method, abstract, use the abstract keyword
  + The sole purpose of the method is to ensure that subclasses will override the method
  + A syntax error will appear if an abstract method is not overridden
* Substitution Principle
  + You can always use a subclass object when a superclass object is expected
  + Instance method calls are always determined by the type of the actual object
    - Not the type of variable containing the object reference
    - Objects of different classes within an inheritance hierarchy can be treaded in a uniform way (polymorphism)
  + Object reference determines what methods an object can call
  + Type of object determines how the method will be implemented
* Upcasting
  + Casting an object of a subclass to a superclass reference
  + Undergraduate\_V3 undergrad1 = new Undergraduate\_V3();  
    Graduate\_v3 graduate1 = new Graduate\_v3();  
    Student\_v3 student1 = undergrad1
  + Cast is OK because an object of a subclass is always an object of its superclass
* Downcasting
  + Casting an object of a subclass to a subclass reference
    - To call a method in a subclass that does not exist in the superclass
  + Cast required since java needs confirmation you understand you are downcasting
  + Downcasting is impossible if objects are not related
  + ClassCastException thrown if object being cast is not compatible with the new type it is being cast to
  + instanceOf
    - it is a good idea to confirm you can perform the downcast
* Polymorphism
  + Many shapes
  + Refers to the fact you can
    - Have different subclasses as objects of a single class
    - Automatically select the proper method to apply based on the subclass it belongs to
* Dynamic Binding
  + Calls to overridden methods are resolved at execution time, based on type of object referenced
* Method only in superclass
  + Belong to the superclass
  + Super and sub class object could use the method
* Method in superclass and subclass
  + Polymorphism- the method that runs is determined at run time(object type)
* Method in subclass only
  + Only objects of subclass can use the method
  + Downcast is required
* What are Interfaces
  + Interface is a collection of abstract methods and constants
    - Used to establish a set of methods that a class will implement
    - Provides for consistency of activity across disparate classes
* Interfaces vs classes
  + All methods in an interface are abstract
    - Have name, parameters, and return type, but no implementation
    - All methods are automatically public
    - Cannot have instance variables
    - Cannot have static methods
  + Class can only inherit from one superclass
    - Classes may implement several interfaces
* When to use an abstract class instead of an interface
  + When the super sub class relationship is a “is-a” relationship
* When to use an interface instead of an abstract
  + If a subclass needs to inherit from additional classes
  + If you need to force the creation of similar behavior across disparate classes
* Relationships between classes
  + Generalization/specialization(inheritance)
    - A person “is a” student
  + Association
    - Conceptual relationship, but with no owner
    - Two classes are connected together conceptually in some way, but without ownership
    - A student “uses a” library
    - 
  + Aggregation – weak association
    - Conceptual relationship with an owner
    - Special form of association where two classes are connected together conceptually and have ownership
    - A teacher “has-a” department
    - 
  + Composition – strong association
    - One class owning another class
    - Classes are not indepented of eachother
    - A house “has a” room
    - 
* Types of errors
  + Syntax(compiler) errors
* Logic Errors
  + Actual output does not match expected output
* Runtime Errors
  + Illegal operation/impossible to execute
  + Detected during program execution
  + Treated as an exception in java
* Exception Handling
  + Exit the program – unfriendly
  + Ignore the exception – poor design
  + Deal with exception and continue
    - Print error message
    - Request new data
    - Retry action
* Unchecked exception
  + Represent defects in the program
  + Subclasses of RunTimeException
  + Method is not obliged to establish a policy for unchecked exceptions thrown
* Checked exception
  + Represent invalid conditions outside immediate control of program
  + Subclasses of exception
  + Method must establish a policy for checked exceptions thrown (catch or declare)
* Exception keyword
  + Try
  + Throw
  + Catch
  + Finally
* Procedure for using exceptions
  + Try block
  + Throw to generate exception
  + Catch to specify exception handlers
  + Finally to specify actions after exception
* If exception is uncaught, execution stops
* Stream is an object that manages the data flow bwteen the program dn I/O devices
* Data flow into the program is an input stream
* Data flow out of the program is an output stream
* JFileChooser can be used to choose a file
* FileOutputStream can be used to write text to files
* Scanner class can be used to read from the file
* Text files
* Binary files