1 Definitions of Stuff

Object-oriented programming Object-oriented programming is a programming paradigm based on the specification, implementation, and use of objects.

Object An object is a data type with structure and state. Each object defines methods that can access and manipulate the state. These methods exist to be called by other objects.

Class A class is a structure representing an entity that contains fields and methods relating to this entity. An object is a specific instance of a class.

Information Hiding Information hiding is the concept of hiding implementation details, and making certain fields and methods of your object inaccessible.

Encapsulation Encapsulation is the combination of grouping related data and operations in an common object together with hiding the implementation of this object from its users.

Inheritance Inheritance is the mechanism used to implement hierarchical relationships.

Polymorphism Polymorphism is allowing an reference type to refer to objects of other, different reference types.

Public The `public` modifier allows a method/field to be inherited and accessed from outside the class

Private The `private` modifier prohibits a method/field from being inherited or accessed from outside the class

Protected The `protected` modifier allows a method/field to be inherited but prohibits it to be accessed from outside the class

Static The `static` modifier limits the number of instances of this method/field to one and allows access without an object. I.e. the one instance is shared by all object instances (think: one mailbox!)

Final The `final` modifier prevents subclasses from overriding or changing this method/field

Reference Type A reference type is an address of an object, i.e. it is a value representing the location in memory where we can find a specific object. This is used by Java to pass around objects, instead of passing the whole object around itself.

Base Class The base class is the class on which an inheritance relationship is based. I.e. it is the class from which the other classes derive their properties. It is also called the superclass.

Derived Class A derived class is a class that inherits properties and methods from a base class. It is also called a subclass. It can extend the base class' functionality by adding its own fields and methods, and it can change the base class' functionality by overriding methods.

Super Suppose the class `C` is a class that inherits from class `A`. If used in class `C`, the keyword `super` refers to the superclass of `C`, i.e. `A`.

This Suppose we have an instance `p` of class `Person`. Within the instance `p`, the keyword `this` refers to the instance `p`.

Abstract Method An abstract method is a method that declares functionality that all derived objects must eventually implement.

Abstract Class An abstract class is a class containing an abstract method. It cannot be instantiated.

Interface An interface is an extreme abstract class — it's a class where NO methods are implemented, but only declared.

2 Advantages of Stuff

2.1 Advantages of OOP

1. the user doesn't have to worry about implementation details — they can assume that the object performing the task will do it correctly
2. the programmer doesn't have to worry about telling the user about minute changes to the implementation
3. it organizes the code into chunks that better represent real-life objects, and therefore is logically easier to understand program flow
4. it places related code all in one place, making the program easier to write, read, debug, and maintain
5. it promotes code reuse

2.2 Advantages of Information Hiding

1. the user cannot see or modify the inside of the object, preventing malicious/stupid things from happening (for example, Microsoft doesn't exactly allow its users to modify the code to their MS applications, do they?)

2.3 Advantages of Encapsulation

1. it puts all related code together in one spot. Makes the code easier to write, read, debug, and maintain.
2. divides the code up into logical entities that mimic real-life objects.

2.4 Advantages of Inheritance

1. allows you to extend the functionality of a class without having to rewriet the code
2. logically organizes your code into real-life relationships
3. allows you to write more generic code (i.e. a method that takes in an object of the supertype can also be used on objects of all subtypes)

2.5 Advantages of Polymorphism

1. can write more flexible code without having to worry about exact type of object
2. allows you to write more generic code

2.6 Advantages of Abstract Classes

1. allows you to guarantee functionality to a user of a class, but lets you put off the implementation for awhile

2.7 Advantages of Interfaces

1. makes multiple inheritance possible!
2. allows you to write more generic code

3 Facts About Stuff

- The three main principles of OOP are:
  1. information hiding and encapsulation
  2. polymorphism
  3. inheritance
- An object is an atomic unit. So the user should not want to see or modify the inside of the object.
- Encapsulation can be thought of as a 'has-a' relationship, whereas inheritance is thought of as an 'is-a' relationship.
- The fields of a class should be the properties of the object, and the methods of the class should be ways to modify and access the values of these properties.
- In Java, we implement information hiding using the modifiers `public`, `private`, and `protected`.
- In Java, a class can extend only one other class, but may implement multiple interfaces. Why?
- A class is made up of fields, methods, and constructors.
- A derived class is type compatible with its base class. This means that a reference variable of the base class type can reference an object of the derived class, but *not* necessarily vice-versa!

```
Person p = new Female( "Barbie" );  // OK
Female f = new Person( "Pat" );     // Not OK
```

- You can use `super` to call your subclass' constructor:

```
public Female( String n, int shoes ) {
    super( n );                 // Calls Person's constructor
    pairsOfShoes = shoes;       // Does other Female-specific initializations
}
```

  or to call the subclass' methods:

```
public class Female extends Person {
    // Fields & Other Methods...

    // This will replace the already implemented rateDatingPotential
    // supplied by the Person class.
    public int rateDatingPotential() {
        int p = super.rateDatingPotential();  // First rate as a Person
        p     = p - pairsOfShoes/4;           // A lot of shoes is bad
        p     = p + bustSize/8;               // Rumour: the more bust the better
        return p;
    }
}
```

- There are many ways to use `this`:
  1. You can precede a reference to any field or method in a class by `this` without changing its meaning:
     `this.toString()` ≡ `toString()`.
  2. You can use `this` as an argument, to pass the name `p` as an argument: `bigger( this, e )`

3. You can use `this` to refer to another constructor:

```
public Female( String n, int shoes ) {
    this( n );              // Calls the existing constructor for just a name
    pairsOfShoes = shoes;   // Does other initialization with the other parameter
}
```

- If a class contains an abstract method then the class itself is considered to be abstract. When a derived class doesn't implement all the abstract methods then the methods stays abstract in the derived class, and the derived class itself must remain abstract.
- You cannot instantiate an abstract class because, by definition, not all of the methods exist.
- The methods of an interface are automatically public. An interface, like an abstract class, cannot be instantiated due to its incomplete specification.
- To use an interface we bring in the keyword `implements`. Using `implements` in the class definition means that the class MUST define all of the methods in the interface — it can't leave some blank like implementors of abstract classes can. If you don't define all of the methods then you will get a compilation error.

4 Tricky Stuff

Suppose I have the following classes:

```
public class Banana {
    private    int apple;
    protected  int kiwi;
    public     int mango;

    public Banana () {
        apple = 0;
        kiwi  = 0;
        mango = 0;
    }

   public int getApple()
       { return apple; }

   public int getKiwi()
       { return kiwi; }
}
```

```
public class Rama extends Banana {
    private   int corn;
    protected int peas;
    public    int carrots;

    public Rama() {
        super();    // initializes apple, kiwi, mango to 0
        corn    = 0;
        peas    = 0;
        carrots = 0;
    }

public int getCorn()
    { return corn; }

public int getKiwi()
    { return 5; }
}
```

```
// THIS IS IN ANOTHER FILE
class Test {
     public static void main( int[] args ) {
         Banana b = new Banana();         // OK
         Rama   r = new Rama();           // OK

         Banana n = new Rama();           // OK
         Rama   m = new Banana();         // Not OK

         b.apple;                         // Not OK, apple is private
         b.kiwi;                          // Not OK, kiwi is protected
```

```
        b.mango;                        // OK, mango is public

        r.apple;                        // Not OK, apple is not inherited, nor is it public
        r.kiwi;                         // Not OK.  Although kiwi is inherited to rama, it is pro
        r.mango;                        // OK. mango is inherited, and public

        r.corn;                         // Not OK
        r.peas;                         // Not OK
        r.carrots;                      // OK

        n.mango;                        // OK because n is a Banana
        n.carrots;                      // Not OK because n is a Banana, not a Rama
        ((Rama)n).carrots;              // OK because we actually put a Rama into a Banana

        n.getKiwi();                    // THIS CALLS Rama's getKiwi and returns 5!!
        n.getCorn();                    // Not OK
        ((Rama)n).getCorn();            // OK!
    }
}
```

5 Questions Aboot Stuff

1. Why do we want to information hide?
2. Why do we want to encapsulate?
3. Why do we want to use inheritance?
4. In our Person class:
   - what were the properties of a Person?
   - what were the fields representing?
   - what were the things we wanted to do to a person?
   - what were the methods of a person?
5. How do we determine what an objects' fields are? Its methods?
6. What is the difference between public and protected? private and protected? Why is protected good? Why do we want to hide the fields (why not all public?)
7. What is a static field good for?
8. Name some inheritance hierarchies (examples).
9. How do I do inheritance in Java? What gets inherited?
10. Why can't I use extends with more than one class?
11. Why would we want to override a method?
12. What are the conditions for overriding a method?
13. What is the constructor for? Does it have a return type? What is its name?
14. What keyword/operator do you use to create an object?
15. Do you understand casting and when it has to be used? Write some examples of legal and illegal calls to methods. (a-la Female and Person sort)

16. What is super? What is this? How can I use them?
17. If I can't instantiate an abstract class, what's its point?
18. What do you have to do in order to make a class not abstract if it inherits from an abstract class?
19. How is an interface different from an abstract class? How are they similar?
20. Draw some class hierarchies based on inheritance relationships. Draw them from 'extends' relationships, 'implements' relationships, etc.
21. What methods does the Iterator interface have? What do they do? Why do we have this interface? If I asked you to implement it in some way, could you do it?

6 Suggestions, If You Want 'Em

- Studying this package is not enough! You have to look over the notes, the examples, and some code, too!
- Try answering some questions *on paper*. Your test will be on paper, so you should practice being able to write stuff down without looking at your notes. For example, try to write down some definitions from memory.
- Try writing out sample Java code... you know there will be some on there. Practice writing a class. Practice putting in fields, methods, and constructors. Practice using `extends` and the modifiers `public`, `protected`, and `private` (yuck!). Practice writing recursive methods for anything you can think of. Practice implementing an interface. Write it out!
- Read over your notes often. Don't just read the facts, but read the examples and explanations, too. They may help you remember the facts later!
- Look over your assignments and try to understand what you did, and why you got it right or wrong. Just because you aced the assignment doesn't necessarily mean you still remember what you were doing when you wrote the assignment, unfortunately.
- Relax, don't stress, and remember that you're a smart person! Remember that you can reason your way through questions. And if you don't understand what a question is asking then put up your hand!