Write these classes. Use good code formatting (formatting here is to save space).
If you write all the classes in the same file, then only one class can be "public" and the filename must be the name of the public class, i.e. "public class A" and A.java. Or, put each class in its own file (the usual way) and declare each class as "public".

```
class A {
    public A() {
        System.out.println("constructing a new A");
    }
    // no toString() method yet
}
// B is a subclass of A.  Inherits code from A.
class B extends A {
    public B() {
        System.out.println("constructing a new B");
    }
    public String toString( ) { return "I'm a B"; }
}
class C extends B {
    public C() {
        System.out.println("constructing a new C");
    }
    public String toString( ) { return "C, of course"; }
}
class Main {
    // Please use methods for these problems. Don't pile lots of code into main().
    public static void testConstructors() {
        System.out.println("create an A object");
        A a = new A();
        System.out.println("create a B object");
        B b = new B();
        System.out.println("create a C object");
        C c = new C();
    }
    public static void main(String[] args) { testConstructors(); }
}
```

1.1 What is printed when your code executes `C c = new C();`

1.2 Explain why this happens.

2. Which of the following is legal? (Create a method in Main and try it.)

```
______  A a  = new B( );
______  B b  = new A( );
______  C ca = new A( );
______  A ac = new C( );
```

3. Class A does not have a toString method, but you can call `a.toString()`.

```
public class Main {
    public static void testToString() {
       A a = new A( );
       System.out.println("a = "+a.toString());
    }
    public static void main(String[] args) { testToString(); }
}
```

3.1 Why can we call a.toString()? It must execute code from *somewhere*. Where does the toString code come from?

3.2 Modify testToString to make **a** refer to a C object (problem 2 shows that this is legal).

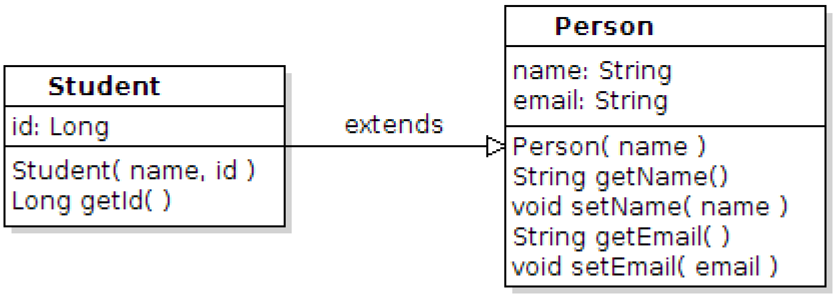
```
    public static void testToString() {
       A a = new C( );
       System.out.println("a = "+a.toString());
    }
```

Now what is printed? Explain **why**. (If you don't know, then *speculate or hypothesize* the reason.)

4. We want to define a Student class with these attributes and methods:

Student
id: Long name: String email: String
<<constructor>> Student(name, id) Long getId( ) String getName( ) void setName(String) String getEmail( ) setEmail(String)

Many of the attributes and methods are the same as Person. To avoid duplicate code (and enable polymorphism) define Student as a subclass of Person (from objects-lab last week):



4.1 Why do you think the Student class does **not** have a setId( ) method?

4.2 Write your code in the **objects-lab** project from last week. Define a Student class in the person package.

```
// Student.java
package person;
/**
  * TODO: Write javadoc.
  */
public class Student extends Person {
    private long id;
    public Student(String aname, long id) {
        setName(aname);  //Error: why?
        this.id = id;
    }
    public String toString() {
        return String.format("Student %s [%d]", getName(), id);
    }
}
```

4.2 Why do we get an error in the Student constructor? How to fix it?

4.3 After fixing the Student constructor, run this method to verify that a Student can use methods of Person, even though you didn't write them in the Student class. Put the code in a "Main" class, not the Student class.

```
    public static void testStudentExtendsPerson() {
        Student std = new Student("Fatalai", 6010540000L);
        System.out.println( "name: " + std.getName() );
        System.out.println( "email: " + std.getEmail() );
        System.out.println( "id: " + std.getId() );
    }
    public static void main(String[] args) {
        testStudentExtendsPerson();
    }
```

4.4 Both Student and Person have a toString() method. What determines which method is invoked? Run:

```
    public static void testDynamicToString() {
        Person p1 = new Person( "Peewee");
        System.out.println( "p1 = " + p1.toString() );
        Person p2 = new Student( "Peewong", 100L );
        System.out.println( "p2 = " + p2.toString() );
    }
```

You can also see this using classes from the Java API.

```
    public static void testToString() {
        Object obj = new java.util.Date();
        System.out.println("obj = "+obj);
        // same variable!
        obj = new Double(Math.PI);
        System.out.println("obj = "+obj);
    }
```

5. How can you set a student's email address in this code? How can you print the email?

```
    public static void testCreateStudent() {
    Student student = new Student("Fatalai Jon",  6010540001L);
    String email = "fatalai.j@ku.th";
    _______________________________;  // how to set student's email?
  // print id, name, and email
  System.out.printf("id: %d  name: %s  email: %s \n",
                 ___________, ____________, ________ );
    System.out.println( p.toString() );
```

6. What happens if Person and Student *both* have an "id" attribute?

```
public class Person {
    private long id; // national id
    public void setId(long id) { this.id = id; }
    public long getId( ) { return id; }
    . . .
}
public class Student extends Person {
    private long id;    // student id
    public long getId( ) { return id; } // return student id
    . . .
}
```

Write some test code to determine what happens. Is it...

(a) illegal (compile error)
(b) Student "id" attribute replaces Person "id" attribute in Student objects
(c) Student has 2 id attributes, but Student's "id" hides (*shadows)* Person "id".

8. How can Student access the Person "id" attribute?

```
public class Student {
    public String describe() {
    return "Student id "+this.id+" and national id "+________________;
```

Guideline: avoid redeifining attributes of a parent class. If you need to redefine an attribute, it often indicates a misuse of inheritance or bad design.

The *Liskov Substitution Principle* (LSP)

The *Liskov Substitution Principle* is a criterion for correct use of inheritance.

If a type *S* is a subtype of type *T* (i.e. *S* is a subclass of *T*), then, in any program that uses objects of type *T,* we should be able to *substitute* type *S* objects in place of type *T* objects and the program will still function correctly.

For a method this means: if we have a method `f(T)` that accepts an object of type *T*, the method should work if we pass it an object of type *S*: `f(s)` where s is type *S*. For example, System.out.println(Object) should work correctly with an object from a subclass of Object.

Apply the Liskov Substitution Principle below.

Java has 4 access levels: **`public`**, **`protected`**, *default* (package access), and **`private`**. The *Java Tutorial* and *Big Java* explain their meaning in detail.

Access Modifier	Member can be accessed from:
`public`	anywhere
`protected`	this class, a subclass, or any class in the same package
`default` (*no modifier)*	this class or any class in same package
`private`	this class only

9. Use this table and the LSP to explain how a subclass *should* be able to change access level (visibility) of a method that it overrides. Using Person and Student (subclass) as example, if both Person and Student have a method named foo( ), and our code does this:

```
class Main {
    public static void accept( Person person ) {
        person.foo( );
   }
```

LSP says this method should work if we call `accept( student )` using a Student instead of Person.

For inheritance and LSP to work, the **accept** method *must* be legal if we give it a Student object as parameter. That is:

```
Student student = new Student("Fatalai", 6010540000L );
Main.accept( student );
```

The accept method must be able to invoke student.foo( ), because it was able to invoke person.foo( ). Using that, in the Student class what access values can we put on the foo( ) method?

Visibility in superclass (Person)	Allowed access (visibility) for Student foo( ) method:
`public void foo( )`	
`protected void foo( )`	
`void foo( )` (*no modifier)*	
`private void foo( )`	

9.2 Which foo( ) method will be invoked by the accept method? Run this code. Explain the answer.

```
public class Person {
    public void foo() { System.out.println("Person foo"); }
}
public class Student extends Person {
    public void foo() { System.out.println("Student foo"); }
}
public class Main {
    public static void accept(Person p)  { p.foo( ); }

     public static void main(String [] args) {
         Student s = new Student("Harry Potter", 6010540002L);
         accept( s );
}
```

9.3 In Person and Student, make the **foo** method **static**. Run the Main class again. Which foo method is invoked? Can you explain why?

10. The Object class defines a **clone** method that returns a copy of an object. It is **protected**.

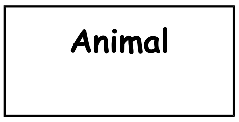
```
  protected Object clone();     // clone is protected in Object
```

In Person we write a clone() method to *override* Object.clone(). What access (public, protected, default, or private) can we assign to the method? Use LSP.

```
public class Person implements Comparable<Person> {
  __________ Object clone( );  // what access levels are allowed?
```

Visualizing Inheritance

1. Assign these classes and methods to an *inheritance hierarchy*. Show your answer as a UML class diagram. Animal should be at the top.



Animal

Cat

Dog

Tiger

Lion

Wolf

Bird

Feline (the Felidae family, animals related to Cats)

Canine (the Canidae family, includes dogs, wolves, coyotes, foxes)

Parrot

Eagle

howl( )

climbTree( )

eat(Food)

fly()

Coyote

purr( )

Bat

Note: Bats can fly, but they are <u>not</u> a kind of Bird!

2. We want to **<u>play</u>** with domestic animals (cat and dog) but not with tigers, wolves, etc. How can we indicate that certain types of Animals should have a play( ) behavior?

What is the best way to model this?

3. Birds and Bats can fly, but a bat is <u>not</u> a bird. How should we design **fly** behavior?

4. Which classes represent *families* of species, but not an actual kind of animal (has no instances)? Write <<abstract>> on those classes.

=== OLD PRACTICE PROBLEMS (optional for extra practice) ===

5. Our code creates a Bird like this:

```
Animal a = new Eagle( );
```

(a) What methods can we invoke using only the **a** reference? (Don't include methods from Object.)

(b) What methods of Eagle can we <u>not</u> invoke using the **a** reference?

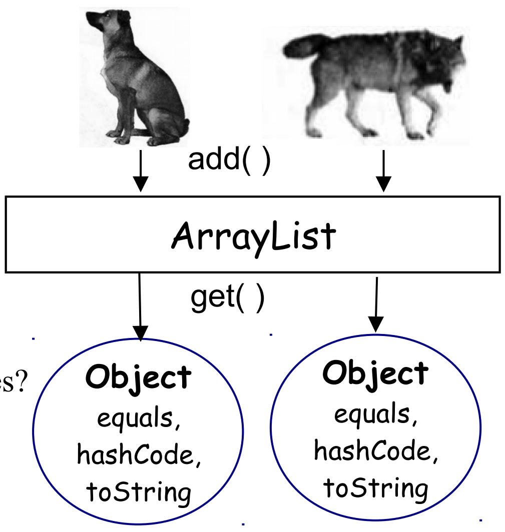
(c) a.fly() doesn't work even though a *refers* to a bird.. How can we tell a to fly?

6. Which of these is not legal?

```
Animal a = new Animal( );
Animal b = new Wolf( );
Canine c = new Coyote( );
Dog d = new Coyote( );
```

7. When we put Canines in an ArrayList, they come out *looking like* Objects. They don't howl any more.

```
ArrayList list = new ArrayList( );
list.add( new Dog() );
list.add( new Wolf() );
list.add( new Coyote() );
x = list.get( 0 )  ---> returns an Object
x.howl( )  // ERROR
```

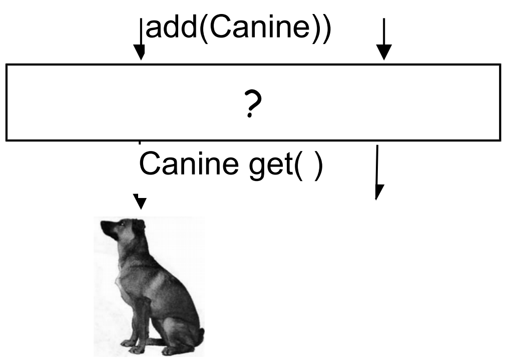


How can we fix this so they come out of ArrayList as canines?

We want:

```
Canine x = list.get( k );
x.howl ();
```

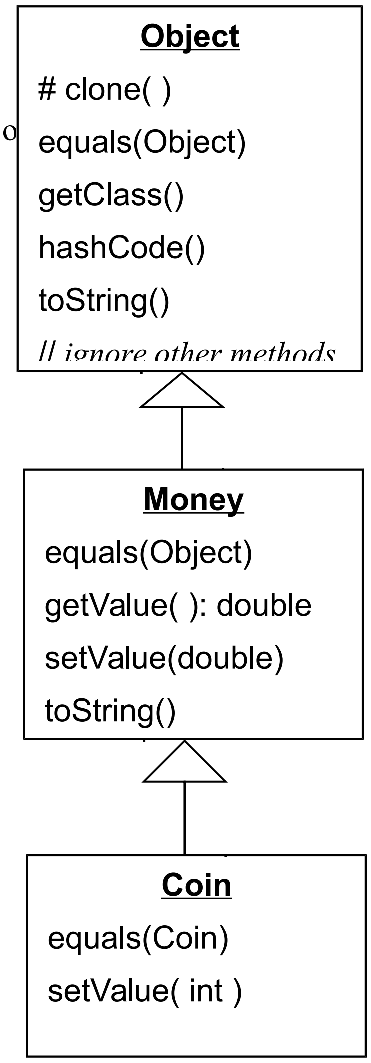
Solution:



8. Suppose we create a coin reference like this:

Coin coin = new Coin(10);

(a) list all the methods that we can invoke using coin (ignore the methods o Object which are not shown in this diagram).



clone( )	from Object
equals(Object)	from Money
getClass( )	from Object
hashCode()	from Object
toString( )	from Money
getValue()	from Money
setValue(double)	from Money
setValue(int)	from Coin A NEW METHOD

(b) the value of Money is double but the value of a Coin is an int.

Can Coin define a getValue( ) method that returns int? Wny?

(c) Suppose that money.getValue( ) returns double and coin.getValue() returns int. Show how this would violate the substitution principle. Hint: consider a method that has a parameter of type Money and calls getValue(). What happens if we invoke this method using a Coin as parameter value?

(d) We want to write a clone( ) method for Coin. What access level (visibility) can we use for clone?

private, package (default), protected, public