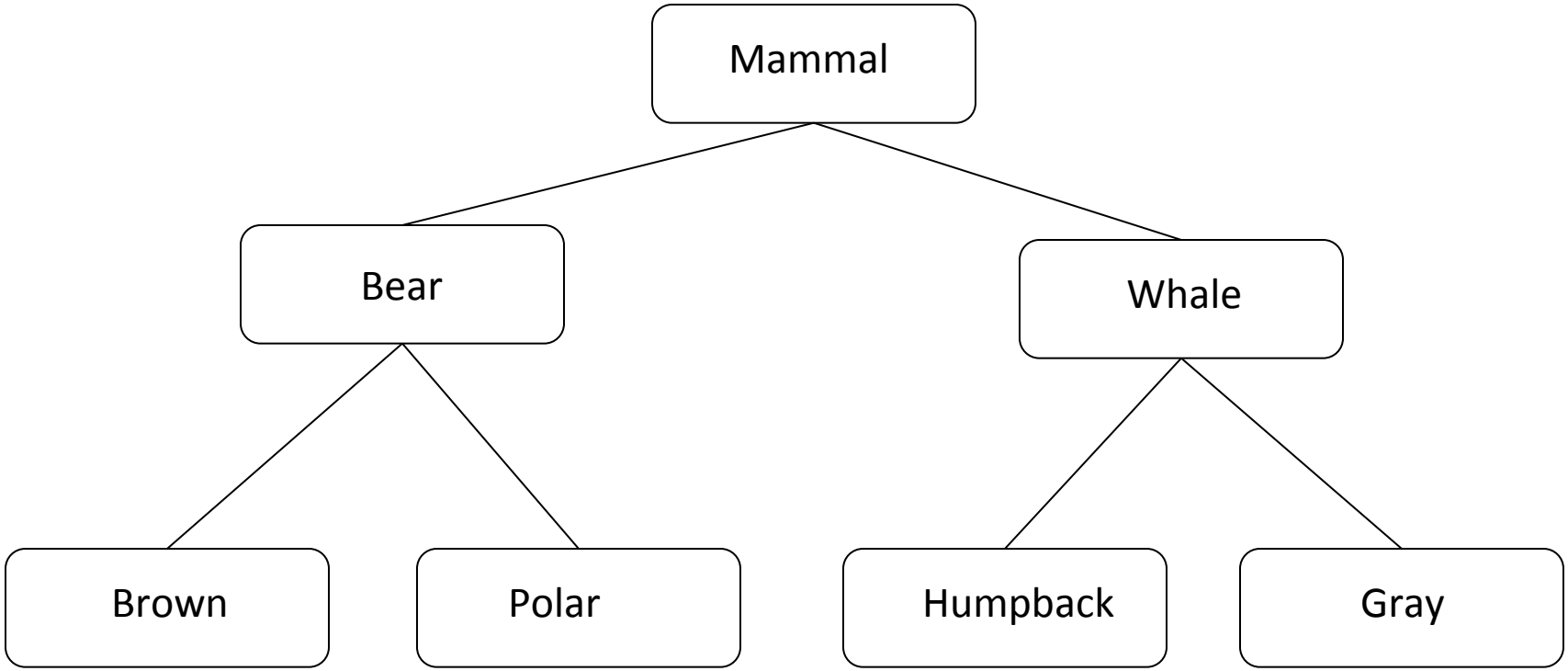
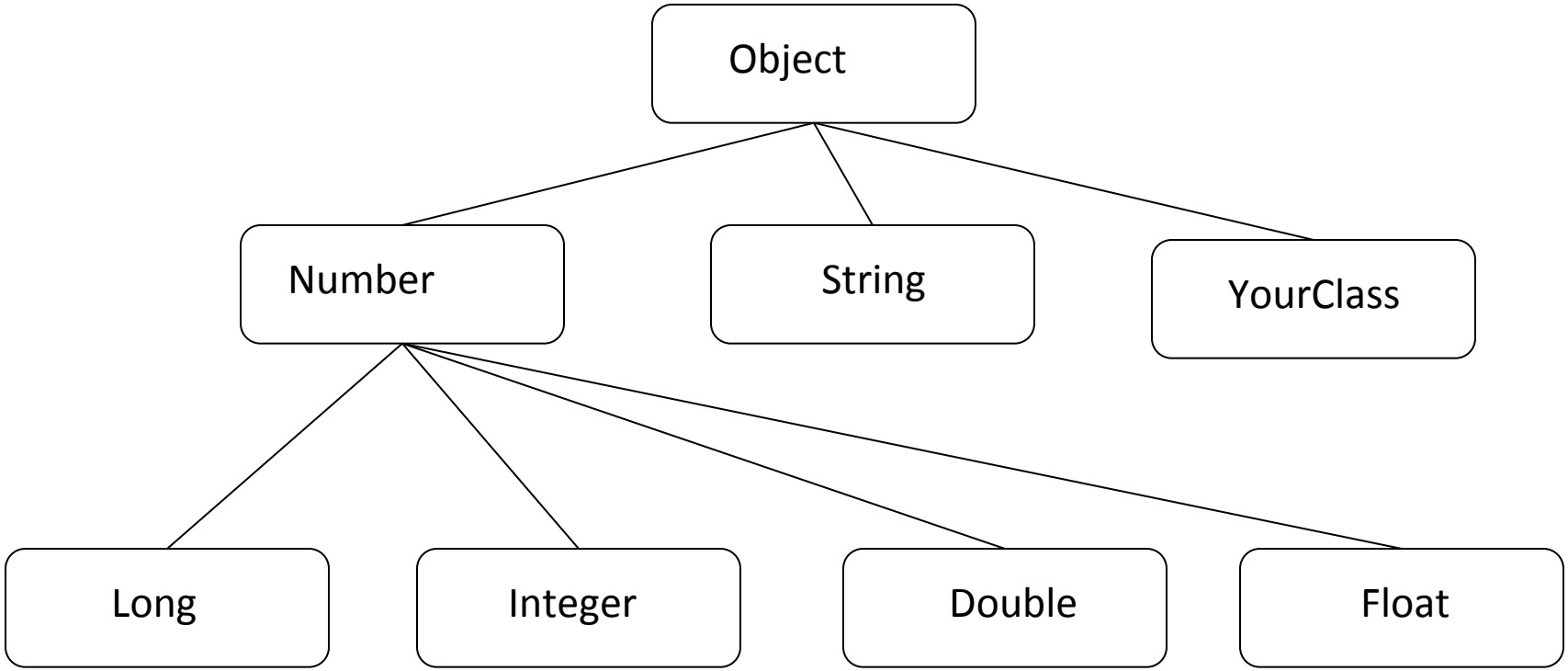
Inheritance

Ch. 10

An example from Zoology



An example from java.lang



Definitions

- Inheritance − classes can be derived from other classes, thereby *inheriting* fields and methods from those classes.
- An object of the derived class, or **subclass**, has access to all public members of the derived class as well as the public members of the **superclass**.

```
//Person.java - characteristics common to all people
public class Person {
   private String name;
   private int age;
   private char gender;
   Person(String name)             { this.name = name; }
   void setAge(int age)            { this.age = age; }
   void setGender(char gender)     { this.gender = gender; }
   void setName(String name)       { this.name = name; }
   int getAge()                    { return age; }
   char getGender()                { return gender; }
   String getName()                { return name; }
   public String toString() {
      return ("Name: " + name +
      ", Age: " + age + ", Gender: " + gender);
   }
}
```

```
//Student.java - an example subclass
public class Student extends Person {
   private String college = "Unknown";
   private int year; // frosh, sophomore,
   private double gpa; //0.0 to 4.0
   Student(String name)                { super(name); }
   void setCollege(String clg)         { college = clg;}
   void setGpa(double gpa)             { this.gpa = gpa; }
   void setYear(int year)              { this.year = year; }
   String getCollege()                 { return college; }
   double getGpa()                     { return gpa; }
   int getYear()                       { return year; }
   public String toString() {
      return(super.toString() + "\n  " +
         "College: " + college +
         ", GPA: " + gpa +
         ", Year: " + year);
   }
}
```

```
//StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student student = new Student("Diane Nguyen");
        student.setAge(21);
        student.setGender('F');
        student.setCollege("UCSC");
        student.setYear(1);
        student.setGpa(3.75);
        System.out.println(student.toString());
    }
}
```

```
//StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student student = new Student("Diane Nguyen");
        student.setAge(21);
        student.setGender('F');
        student.setCollege("UCSC");
        student.setYear(1);
        student.setGpa(3.75);
        System.out.println(student.toString());

        Person p = student;
        p.setGpa(4.0); // NOT ALLOWED – see next slide
    }
}
```

Subtype Principle

- Methods defined in one class may be redefined in a subclass. This is called **method overriding**.
- *A subclass object can always be used where an object of its superclass is expected.*
- Treating a subclass object as a superclass object can only remove capabilities, not add them. With inheritance, new methods are added in the subclass, never taken away.

Overriding vs Overloading

- **Overloading** is when you define two methods with the same name, in the same class, distinguished by their signatures.
- **Overriding** is when you redefine a method that has already been defined in a parent class (using the exact same signature).
- Overloading is resolved at compile time.
- Overriding is resolved at runtime.

Important to remember that overriding only applies to methods, not variables.

Dynamic Method Dispatch

- Determing at runtime, which overriden method to call, is called *dynamic method dispatch*.
- This is what allows println() to work with any object.
- toString() is defined in Object (the parent of all classes).
- If you override toString(), then inside of println(), a call to printThis.toString(), will get to YOUR toString().

```
//StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student student = new Student("Diane Nguyen");

        student.setAge(21);
        student.setGender('F');
        student.setCollege("UCSC");
        student.setYear(1);
        student.setGpa(3.75);
        System.out.println(student.toString());

        Person anyPerson = student;
        System.out.println(anyPerson.toString());
    }
}
```

Which toString() gets called? Recall, both Student and Person classes have a toString() method with an identical signature.

A) toString() from Person

B) toString() from Student

```
//StudentTest.java
public class StudentTest {
    public static void main(String[] args) {
        Student student = new Student("Diane Nguyen");

        student.setAge(21);
        student.setGender('F');
        student.setCollege("UCSC");
        student.setYear(1);
        student.setGpa(3.75);
        System.out.println(student.toString());

        Person anyPerson = student;
        anyPerson.setYear(3);
    }
}
```

Is the bolded line legal?

```
public class ClassOne {
   int data = 101;
   public int get() {
      return data;
   }
   public int mystery() {
      return get()*10;
   }
}

public class ClassTwo extends ClassOne {
   int data = 202;
   public int get() {
      return data;
   }
}
```

```
public class OneTwoTest {
   public static void main(String[] args) {
      ClassOne one = new ClassOne();
      ClassTwo two = new ClassTwo();
      System.out.println(one.mystery()); // prints 1010
      System.out.println(two.mystery()); // prints what?
   }
}
```

What does the second print statement print?

```
public class ClassOne {
   int data = 101;
   public int get() {
      return data;
   }
   public int mystery() {
      return get()*10;
   }
}
public class ClassTwo extends ClassOne {
   int data = 202;
   public int get() {
      return data;
   }
}
```

```
public class OneTwoTest {
   public static void main(String[] args) {
      ClassOne one = new ClassOne();
      ClassTwo two = new ClassTwo();
      fiddle(one); // prints 1010 then 101
      fiddle(two); // what does this print?
   }
   public static void fiddle(ClassOne one) {
      System.out.println(one.mystery());
   }
}
```

What does the second **fiddle** statement print?

```
public class ClassOne {
   int data = 101;
   public int get() { return data; }
   public int mystery() {
      return get()*10;
   }
}
public class ClassTwo extends ClassOne {
   int data = 202;
   public int get() { return data; }
}
```

```
public class OneTwoTest {
   public static void main(String[] args) {
      ClassOne one = new ClassOne();
      ClassTwo two = new ClassTwo();
      fiddle(one); // prints 1010 then 101
      fiddle(two); // what does this print?
   }
   public static void fiddle(ClassOne one) {
      System.out.println(one.mystery());
      System.out.print(one.data);
   }
}
```

What does the second **fiddle** statement print?

```
public class ClassOne {
   int data = 101;
   public int get() { return data; }
   public int mystery() {
      return get()*10;
   }
}
public class ClassTwo extends ClassOne {
   int data = 202;
   public int get() { return data; }
}
```