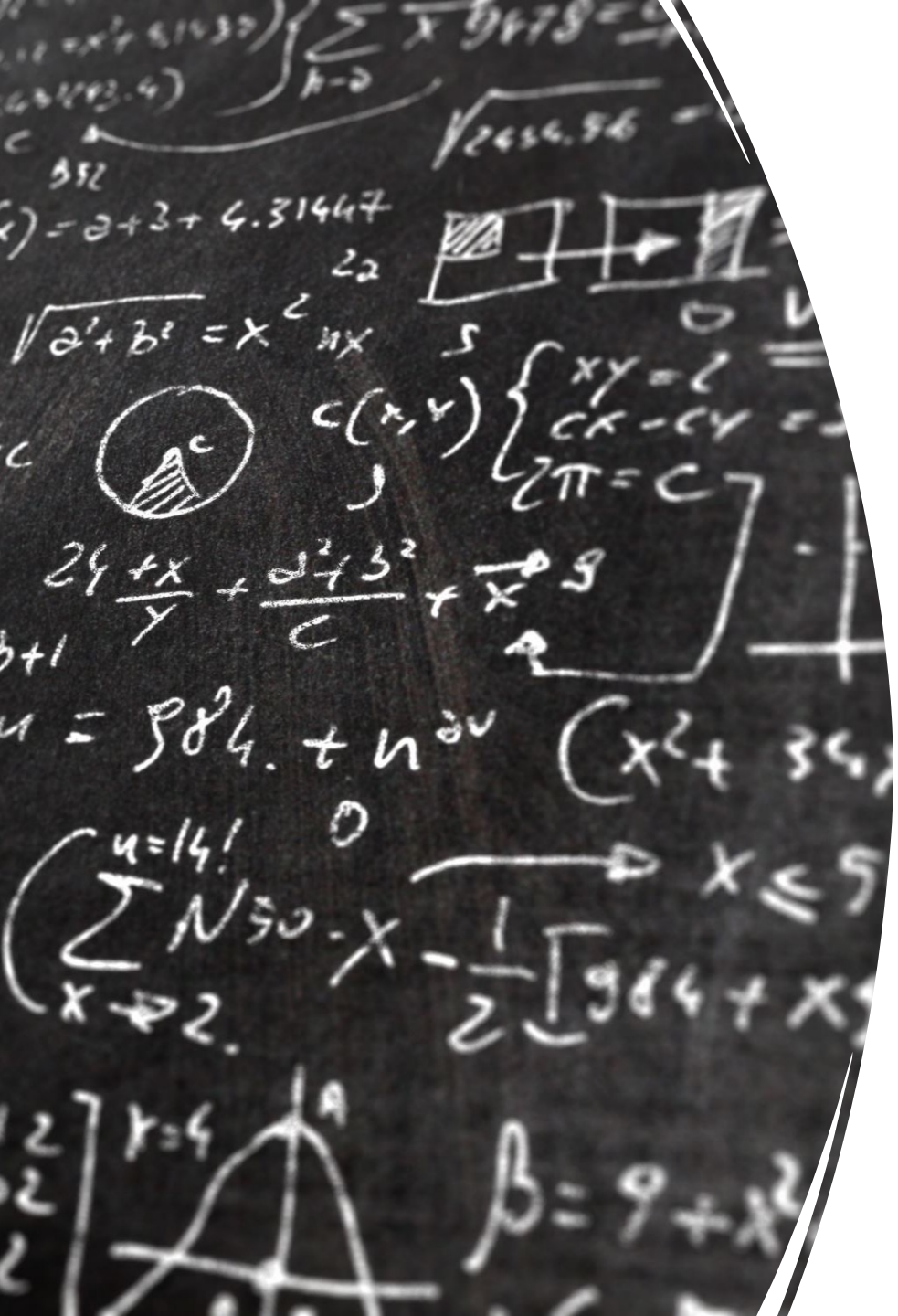


CS 113 – Computer Science I

Lecture 15 – OOP

Adam Poliak

03/16/2023

Announcements

- HW06
  - Due Monday 03/20
  - No autograder
- Midterms:
  - Grades by end of this week
  - Was sick over Spring break
- Week after spring break
  - Hell week – first ten minutes of Thursdays class
- Midsemester feedback

Homework Hints

If we are iterating through an array, which type of loop should you use?

for loop!

We can split a string into an array of strings using .split(…)

Try/catch:

- only use it for code that can throw the specific error you are catching
- Don’t abuse

Object-oriented programming (OOP)

Method for designing programs in terms of objects

Recall: Top-down design

- the “nouns” in your feature list correspond to classes/data
- the “verbs” correspond to methods

Using objects: some special methods

The **constructor method** is called when you do a `new`

accesors (aka getters)
return the values of instance variables

mutators (aka setters)
set the values of instance variables

toString()
returns a string representation of an object

Defining classes

By defining our own classes, we can create our own data types

A class definition contains

- the data contained by the new type (**instance variables**)

- the operations supported by the new type (**instance methods**)

Example: Defining a class `BankAccount`

What data should it have?

- A name
- Amount of dollars

What operations should it support?

- deposit
- withdraw

this

`this` is a special keyword that refers to the object inside an instance method

Analogy:

Visualizing programs with objects

```
class BankAccount {
  public String name = “”;
  public double dollars = 0.0;

  public Point() {
    this.name = “”;
    this.dollars = 0.;
  }

  public Point(String clientName, double money) {
    this.name = clientName;
    this.dollars = money;
  }

 public void deposit(double money) {
     this.dollars = this.dollars + money
}
```

```
  public static void main(String[] args) {
    BankAccount acc = new BankAccount(“Kim”, 0);
    acc.depost(541);

    acc.withdraw(10);
  }
}
```

Draw a stack diagram

Draw a stack diagram

Function Stack:

Created objects

Example: Defining a class `Point`

What data should it have?

- X-coordinate
- y-coordinate
- Name
- color

What operations should it support?

Example: Distance using a static method

- Make a new static function called “add” that takes in two points, adds their x and y coordinates, and returns a new point

Exercise: Objects and Arrays

Arrays can store objects just like any other type (such as ints, Strings, etc.)

Write a program that asks the user for a number of points and stores them in an array.

Exercise: Draw a stack diagram for the previous program

Access modifiers

Specify the access-level of instance variables/methods

- `public`
  - code outside of the class can access the variable/method
- `private`
  - code outside of the class cannot access the variable/method
- `protected`
  - Allow subclasses to accesses data in parent class

Default in java is `public`

Access modifiers

Default in java is `public`

In this class, make instance data private

Class inheritance

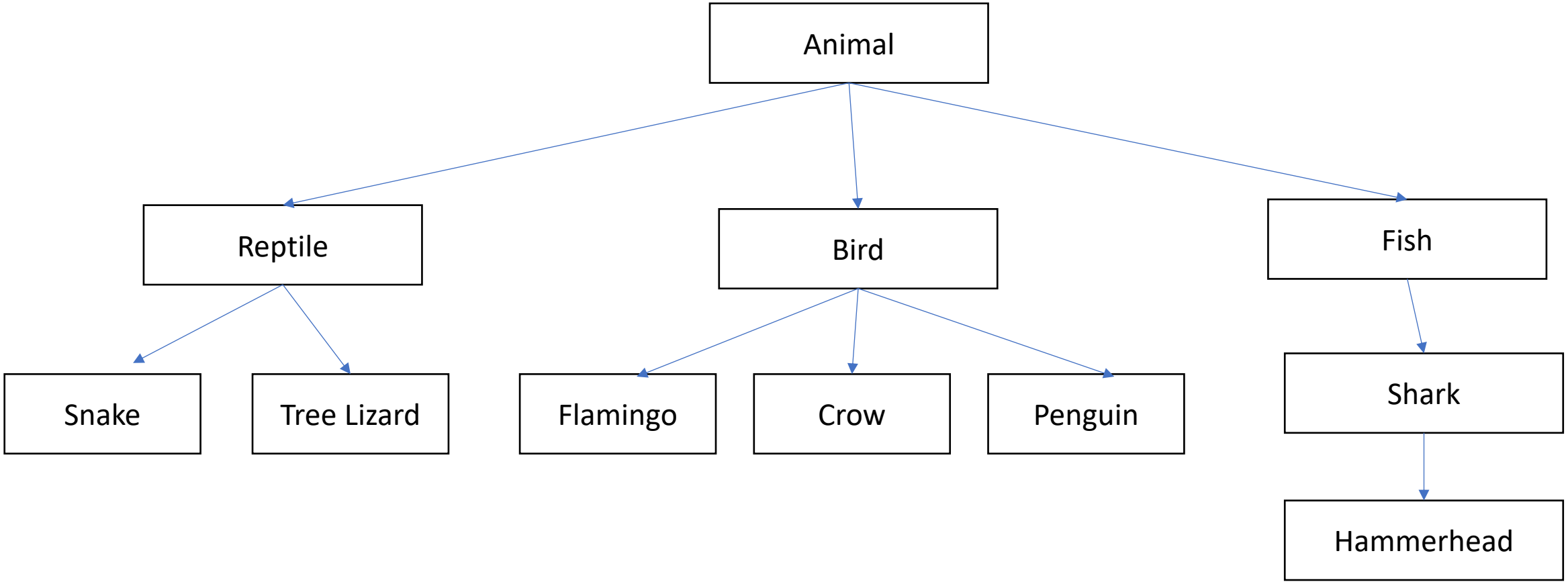
Review:

- Classes are like categories
- Objects are like examples of the categories

Classes can be arranged hierarchically where,

a child class ”inherits” from a parent class

Inheritance: feature for organizing classes into hierarchies



Inheritance: subclasses refine behavior/state

Subclasses can override methods from parent class

Exercise

1. Implement getter functions for instance variables inside Animal

2. In Zoo.java, call the getters and output the values to console

Polymorphism

Program can treat all objects that extend a base class the same

Java automatically calls the specific methods for each subclass

Polymorphism: Demo

```
public class Zoo {
    public static void main(String[] args) {
        Animal animal1 = new Animal();
        animal1.locomote();

        Animal animal2 = new Reptile();
        animal2.locomote();
    }
}
```

```
public class Animal {
    public Animal() {
    }
    public void locomote() {
        System.out.println("I am moving!");
    }
}
```

```
public class Reptile extends Animal {
    public Reptile() {
    }
    public void locomote() {
        System.out.println("I am walking!");
    }
}
```

Exercise: What is the output of this program?

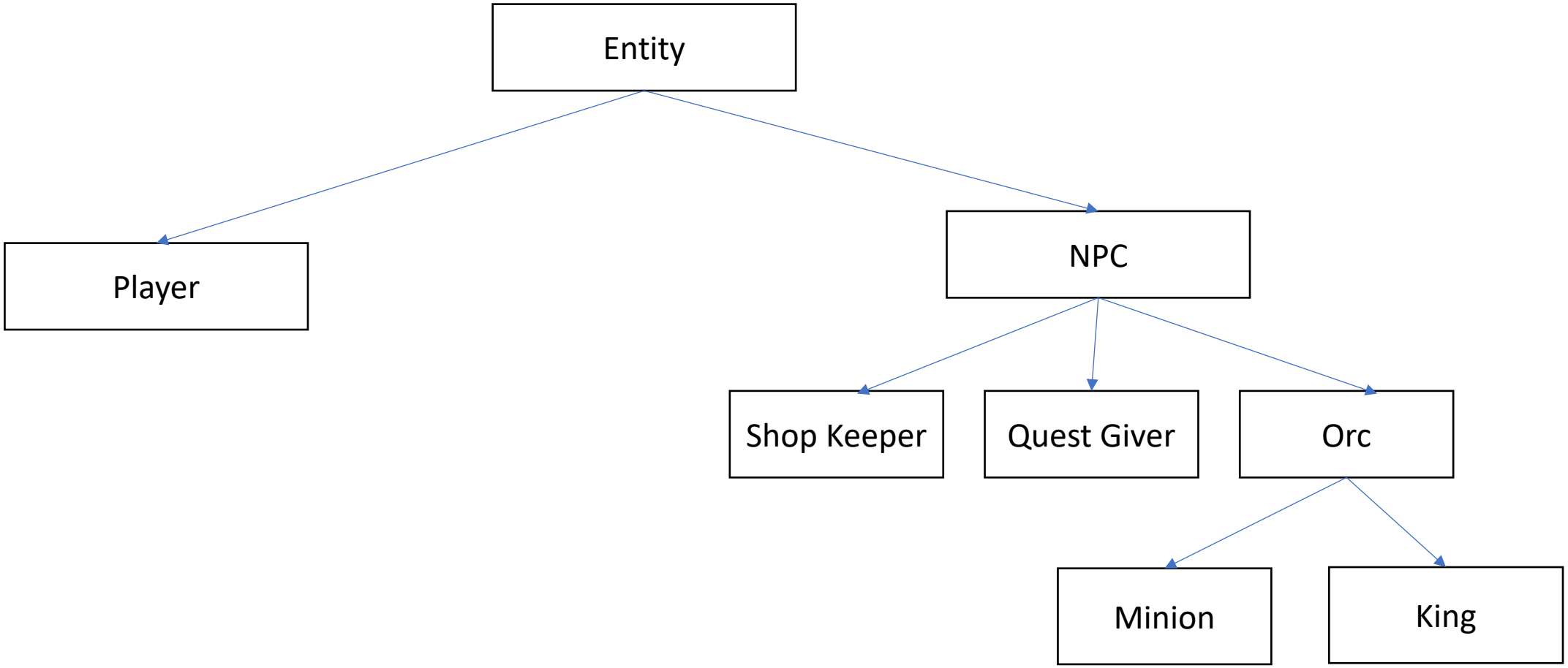
```
public class Zoo {
   public static void main(String[] args) {
      Animal animal1 = new Animal();
      animal1.locomote();

      Animal animal2 = new Fish();
      animal2.locomote();
   }
}
```

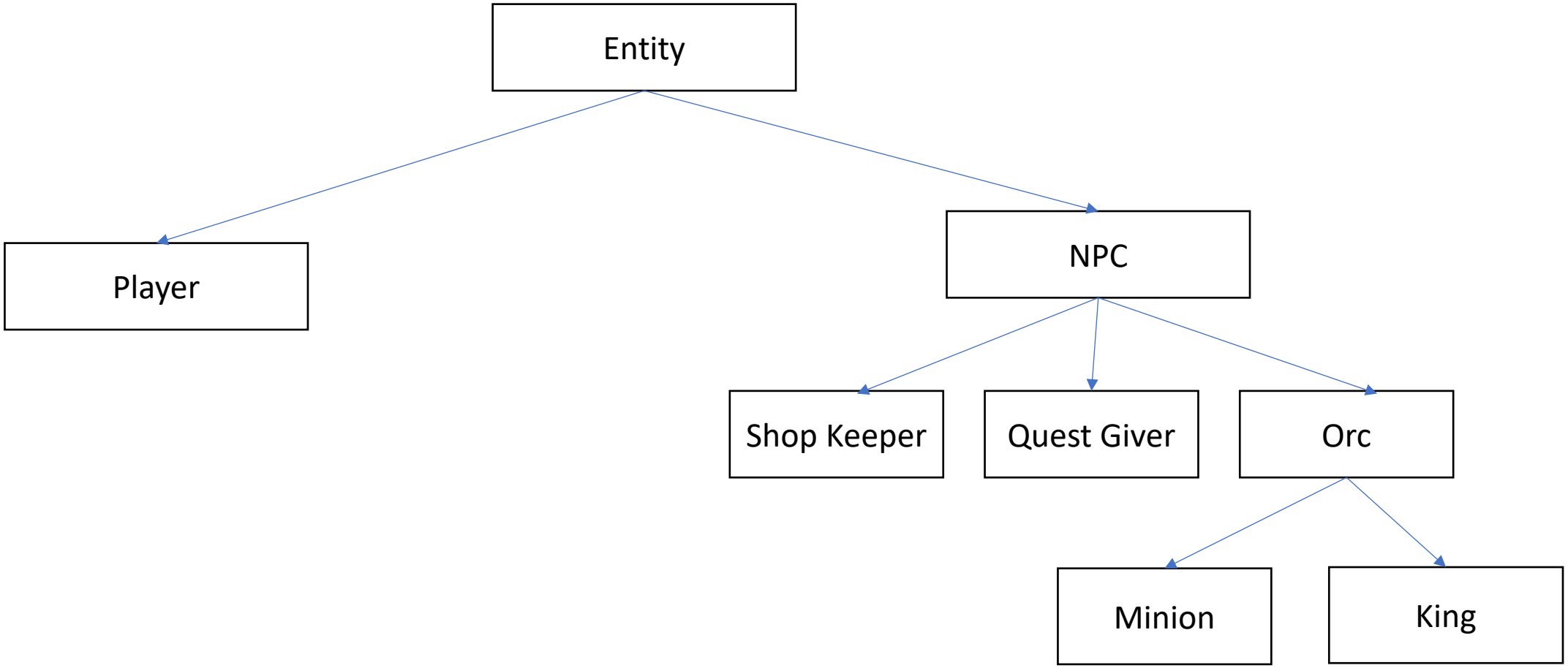
```
public class Animal {
   public Animal() {
   }
   public void locomote() {
      System.out.println("I am moving!");
   }
}
```

```
public class Fish extends Animal {
  public Fish() {
  }
  public void locomote() {
    System.out.println("I am swimming!");
  }
}
```

Question: How would we implement Minion?



Inheritance



Exercise: Implement a Bird animal

OOP Example & Design: Vending machine

OOP Design: Vending machine

Defining the snack class

```
public class Snack {
    private int mQuantity;
    private double mCost;
    private String mName;

    public Snack(String name, int quantity, double cost) {
        mQuantity = quantity;
        mCost = cost;
        mName = name;
    }
    public String getName() {
        return mName;
    }

    public void buy() {
       if (mQuantity > 0) {
           mQuantity--;
       }
    }
}
```

Testing the Snack class

```
public static void main(String args[])
{
    Snack snack = new Snack("Slurm", 10, 1.5);
    System.out.println("Snack: "+snack.getName());
}
```

Objects: Stack diagrams revisited

```
public static void main(String[] args) {
        double userCash = 8.0;
        Snack soda = new Snack("Tang", 10, 1.5); // call constructor
        soda.buy();
}
```

Exercise: draw a stack diagram for this program

Exercise: Define a class BankAccount

BankAccount should have the following data:

- Name
- Amount

BankAccount should have the following operations:

- currentBalance() // returns current amount in the bank account
- withdraw(float amt) // withdraw the given amount from the account
- deposit(float amt) // deposit the given amount to the account