Polymorphism and More Branching

Colorado State University
Department of Computer Science

Slides Originally Created by Marcia Moraes (marcia.moraes@colostate.edu)

Announcements

TODO Reminders:

Readings are due **before** lecture

- Reading 20 (zybooks) – you should have already done that ☺
- Lab 13
- Reading 21 (zyBooks) – you should have already done that ☺
- Lab 14 – optional because of snow day
- Reading 22 (zybooks)
- RPA 10

Keep practicing your RPAs in a spaced and mixed manner ☺

https://www.pinterest.com/pin/342062534189160703/

Wednesday Help Desk – 3-4pm CSB120

Wednesday Help Session – 3-4pm CSB305

Thursday Help Desk – 6-8pm Teams

Thursday Help Session – 3:30-4:30pm Teams

Recall Activity

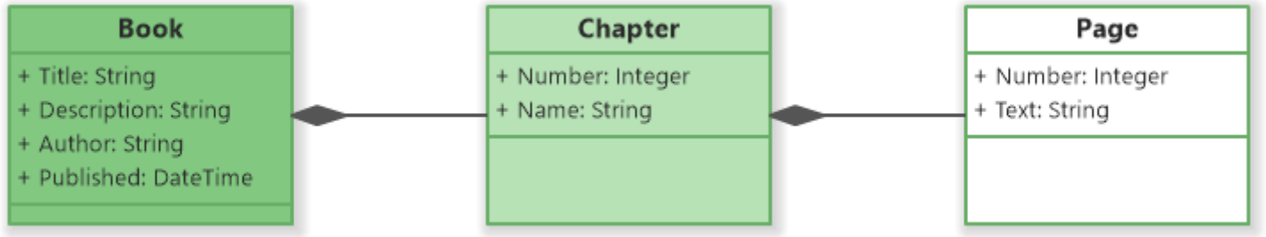
- Analyze the classes below. What type of relationship do we have: ‘has-a’ or ‘is-a’?
- Explain using your own words.

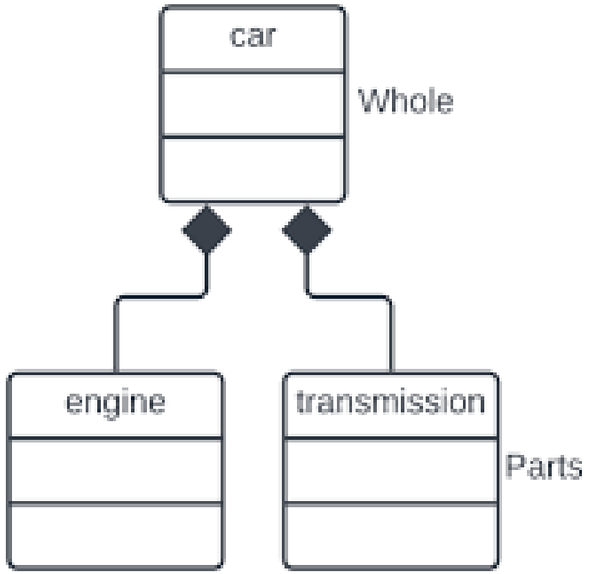
```
public class ChildInfo {
   public String firstName;
   public String birthDate;
   public String schoolName;

   ...
}

public class MotherInfo {
   public String firstName;
   public String birthDate;
   public String spouseName;
   public ArrayList<ChildInfo> childrenData;

   ...
}
```

Composition

- Has-a relationship

```
public class ChildInfo {
   public String firstName;
   public String birthDate;
   public String schoolName;

   ...
}

public class MotherInfo {
   public String firstName;
   public String birthDate;
   public String spouseName;
   public ArrayList<ChildInfo> childrenData;

   ...
}
```

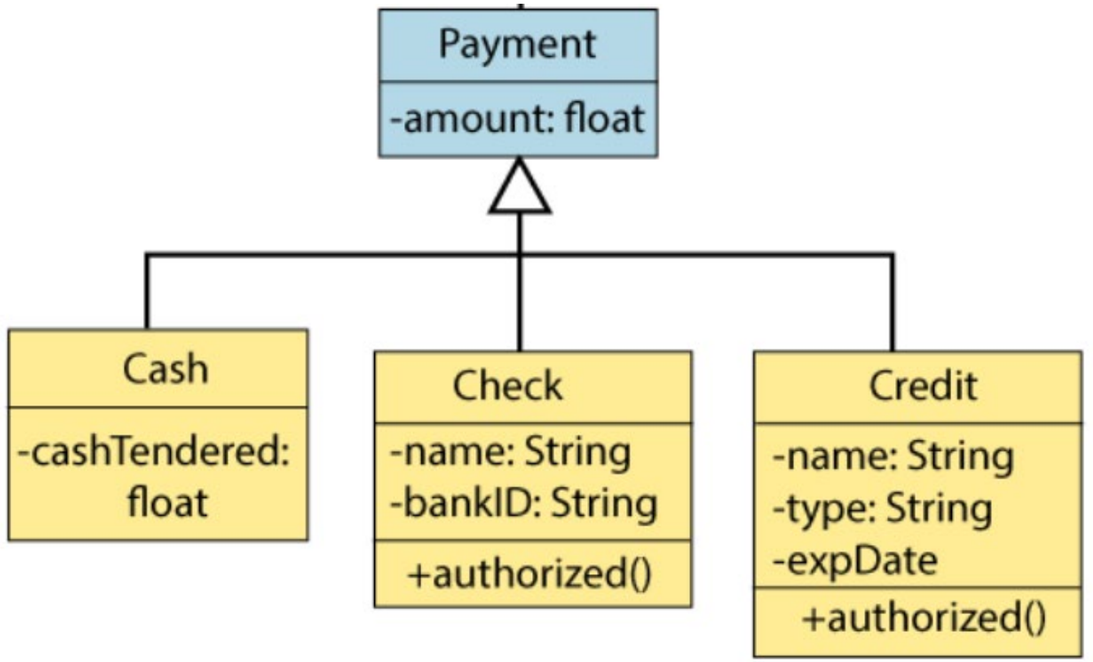


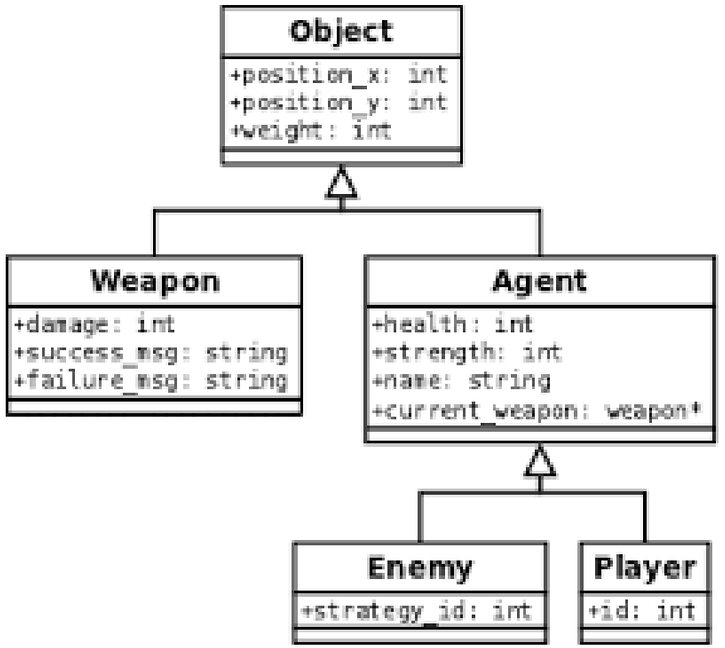


Inheritance

- Is-a relationship

```
public class PersonInfo {
   public String firstName;
   public String birthdate;

   ...
}

public class ChildInfo extends PersonInfo {
   public String schoolName;

   ...
}

public class MotherInfo extends PersonInfo {
   public String spousename;
   public ArrayList<ChildInfo> childrenData;
  ...
}
```





Polymorphism

- Refers to determining which program behavior to execute depending on data types

- Polymorphism of methods – methods overloading
  - **compile-time polymorphism**
  - compiler determines which of several identically-named methods to call based on the method's arguments

- Polymorphism of variables – involves derived classes (inheritance)
  - **runtime polymorphism**
  - compiler cannot make the determination but instead the determination is made while the program is running

Polymorphism of variable

- Substitution principle - you can always use a subclass object when a superclass object is expected
- Super class variable can store super class types and sub class types as well
- Sub class variable can only store sub class types

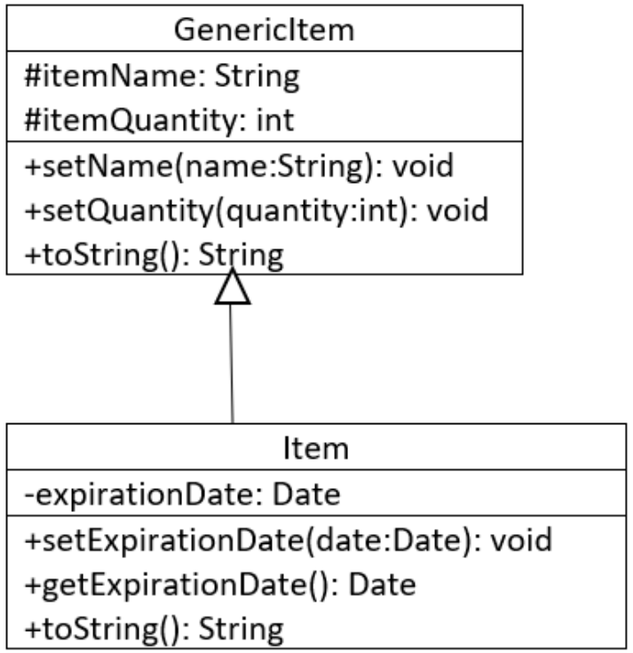
```
GenericItem g1 = new GenericItem();
GenericItem g2 = new Item();
Item g3 = new Item();
Item g4 = new GenericItem();
```

Correct!

Correct!

Correct!

Incorrect!



ArrayList of Objects

- Store a collection of objects of various class types

```
java.lang.Object@4517d9a3
12
3.14
Hello!
ACME -- 5 Main St
```

```
public class Business {
   protected String name;
   protected String address;

   public Business() {}

   public Business(String busName, String busAddress) {
      name = busName;
      address = busAddress;
   }

   @Override
   public String toString() {
      return name + " -- " + address;
   }
}
```

```
import java.util.ArrayList;

public class ArrayListPrinter {

   // Method prints an ArrayList of Objects
   public static void printArrayList(ArrayList<Object> objList) {
      int i;

      for (i = 0; i < objList.size(); ++i) {
         System.out.println(objList.get(i));
      }
   }

   public static void main(String[] args) {
      ArrayList<Object> objList = new ArrayList<Object>();

      // Add new instances of various classes to objList
      objList.add(new Object());
      objList.add(12);
      objList.add(3.14);
      objList.add(new String("Hello!"));
      objList.add(new Business("ACME", "5 Main St"));

      // Print list of Objects
      printArrayList(objList);
   }
```

instanceof

- Rewrite the class Pet to have its constructors properly overloaded.

```
public static void printArrayListV2(ArrayList<Object> objList) {
    int i;
    for (i = 0; i < objList.size(); ++i) {
        Object obj = objList.get(i);
        if(obj instanceof String)
            System.out.println("String:" + objList.get(i));
        else if(obj instanceof Integer)
            System.out.println("Integer:" + objList.get(i));
        else if(obj instanceof Double)
            System.out.println("Double:" + objList.get(i));
        else if(obj instanceof Business)
            System.out.println("Business:" + objList.get(i));
    }
}
```

Conditional Statements/Ternary Statements

```
if(/*condition is true*/) {
    // do something
}else {
    // do something if condition is false
}
```

- A way to write a *simple* if/else on one line.

condition ? value if true : value if false

```
String time = 10 > 5 ? "hello" : "goodbye";

System.out.println(time);// what is printed?
```

hello

Switch Statements

- switches
  - a condition that checks each “case” for using ==
  - concise way to compare against group of options
- case
  - the cases to **==**
- break
  - keeps executing code – until break is called
- Format:

```
switch(primitive or String) {
   case <value>:
       break; //technically optional, but you want it
   default:  // essentially your else
}
```

```
public static String switchTest(String name){
       String faeType;
       switch(name) {
           case "dyson":
               faeType = "Werewolf";
               break;
           case "trick":
               faeType = "Sage";
               break;
           case "bo" :
           case "aife":
               faeType = "Succubus";
               break;
        case "vex":
               faeType = "Mesmer";
               break;
           default:
               faeType = "human";
       }
       return faeType;
    }
```

```
public static void main(String[] args) {
      System.out.println(switchTest("vex"));
      System.out.println(switchTest("bo"));
       System.out.println(switchTest(“kenzi”));
    }
```

Enumerations

- Declares a name for a new type and possible values for that type
- Methods can use them and return them!

```
public enum Names {
   DYSON,
   TRICK,
   BO,
   AIFE,
   VEX,
   KENZIE
}
```

Switch + Enum

- Switch + Enumerations are strong combinations
- Enumeration is part of the case

```
public enum Names {
   DYSON,
   TRICK,
   BO,
   AIFE,
   VEX,
   KENZIE
}
```

```
public static void main(String[] args) {

System.out.println(switchTest(Names.VEX));
System.out.println(switchTest(Names.BO));
System.out.println(switchTest(Names.KENZI));
}
```

```
public static String switchTest(Names name){
       String faeType;
       switch(name) {
           case Names.DYSON:
               faeType = "Werewolf";
               break;
           case Names.TRICK:
               faeType = "Sage";
               break;
           case Names.BO:
           case Names.AIFE:
               faeType = "Succubus";
               break;
           case Names.VEX
               faeType = "Mesmer";
               break;
           default:
               faeType = "human";
       }
       return faeType;
   }
```

Worksheet

- Complete the Polymorphism worksheet

- Codes from this lecture and worksheet Polymorphism - https://github.com/CSU-CompSci-CS163-4/Handouts/tree/main/ClassExamples/10Polymorphism

- Codes from this lecture on switch and enum - https://github.com/CSU-CompSci-CS163-4/Handouts/tree/main/ClassExamples/10MoreBranching

Practice 1 – if..else/switch

- Write an if/else statement that
  - Checks to see if a student_class (String variable) is:
    - "Fencing"
      - Then set meeting_info (String variable) to "Wednesday, 4:30PM"
    - "Boxing"
      - Then set meeting_info (String variable) to "Thursday, "5:00 PM"
    - "Aikido"
      - Then set meeting_info (String variable) to "Monday, 6:00 AM"
    - "Nothing" or null
      - Then set meeting_info to the empty String ("")
- Now write the same statement as a switch statement
- May be easier to write this as two separate methods (if_option, switch_option)

Practice 2 – Enum + Switch

- Write an enum called DiceType
  - Contains d4, d6, d8
- Write a switch statement that takes in the enum:
  - d4 - random number between 1-4
  - d6 – random number between 1-6
  - d8 – random number between 1-8
  - Default – randon number between 1-20
- Remember, Random rnd = new Random(); and rnd.nextInt(6) // returns a range from 0-5
- How do you test this?