[TAP:XBSNJ] Scope

```
public class Baby {
   private String name;
   private int age;

   public Baby(int age, String theName){
      this.age = age;
      name = theName;
   }
   public String toString(){
      return name+ "(age "+age+")";
   }

   public static void main(String[] args){
      Baby baby1 = new Baby(3, "Bill");
      System.out.println(baby1);
   }
}
```

- What does the program output?
  - A. "Bill(age 3)"
  - B. "Bill(age 0)"
  - C. Memory location of baby1 object
  - D. Compiler error
  - E. Whatever

Administrative Details

- Bill is gone! (and will be back at the end of next week)
  - His OH are canceled next week
  - I will take over the classes
  - Dan will take over the labs
- Lab1
  - Due on 2/11/18 (Sun) at 11pm

Agenda

- Inheritance
  - “extends”
    - abstract class
    - interface
    - “implements”

Cookie.java

```
public class Cookie{
  private int calories;
  private boolean isExpired;

  public Cookie(int calories){
    this.calories = calories;
  }  isExpired = false;

  public boolean isEdible(){
    return !isExpired;
  }

  public int getCalories(){
    return calories;
  }
}
```

CookieMonster.java

```
public class CookieMonster{
  private int calories;
  public CookieMonster(){
    calories = 0;
  }
  public void eat(Cookie something){
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat ” + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
  public static void main(String[] args){
  …
  }
}
```

ChocolateChipCookie.java

```
public class Cookie{
  private int calories;
  private boolean isExpired;
  public Cookie(int calories){
    this.calories = calories;
  }

  public boolean isEdible(){
    return !isExpired;
  }

  public int getCalories(){
    return calories;
  }
}
```

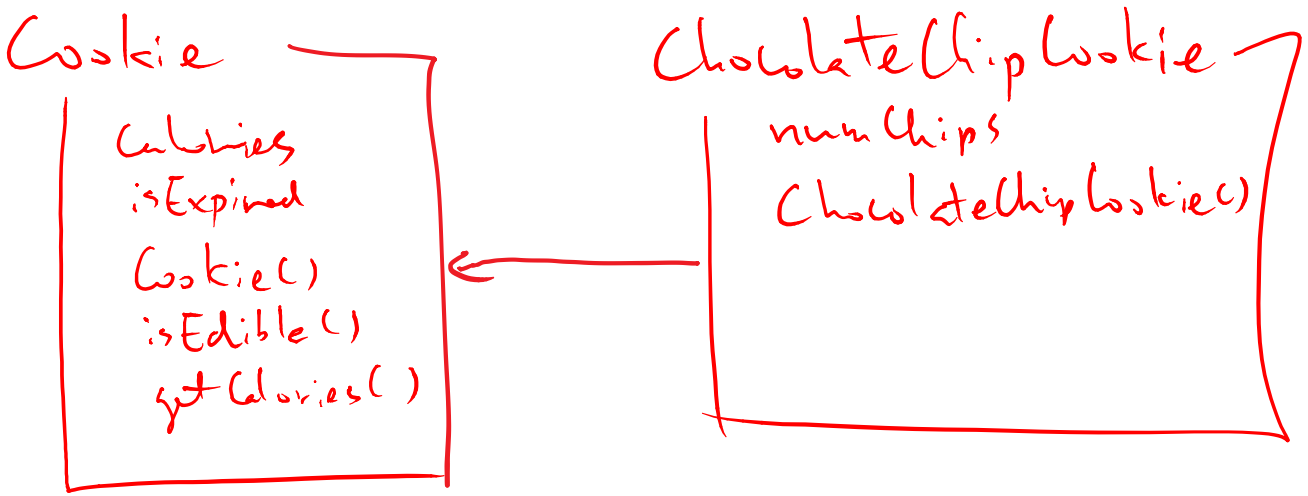
better ChocolateChipCookie.java

child/sub

parent/super

```
public class ChocolateChipCookie extends Cookie{
  private int numChips;

  public ChocolateChipCookie(int numChips){
    super(numChips * 2);
    this.numChips = numChips;
  }
}
```



"extends"

- By inheriting other classes, we can create specialized subclasses, e.g. "is a"

```
public class ChocolateChipCookie extends Cookie{
   …
}
```

- A subclass inherits all non-private fields and methods from the superclass
- An object of a subclass type can be stored in a variable of a superclass type, e.g.

```
Cookie chocoCookie = new ChocolateChipCookie(10);
```

- Note, each class can **extend** up to 1 class

Agenda

- Inheritance
  - “extends”
  - ⊙ abstract class
  - interface
  - “implements”

From the Wiki..

[1][2][3] Despite his voracious appetite for cookies, Cookie Monster shows awareness of healthy eating habits for young children and also enjoys fruits and eggplant.

Apple.java

```
public class Apple{
  private int calories;
  private boolean isChopped;

  public Apple(int calories){
    this.calories = calories;
    isChopped = false;
  }
  public boolean isEdible(){
    return isChopped;
  }
  public int getCalories(){
    return calories;
  }
  public void chop(){
    isChopped = true;
  }
}
```

Orange.java

```
public class Orange{
  private int calories;
  private boolean isPeeled;

  public Orange(int calories){
    this.calories = calories;
    isPeeled = false;
  }
  public boolean isEdible(){
    return isPeeled;
  }
  public int getCalories(){
    return calories;
  }
  public void peel(){
    isPeeled = true;
  }
}
```

Orange extends Apple?

- Orange inheriting Apple (or vice versa) is **not** appropriate.
  - Logically, Orange “is not an” Apple
  - More practically, we don’t want isChopped nor chop() in Orange
- Solution:
  - Create an abstract class Fruit that implements common functionality (methods)
    - if the given method’s implementation is different, declare the method without a body
  - Have Apple and Orange inherit Fruit

Fruit.java

```
public abstract class Fruit{
  private int calories;

  public Fruit(int calories){
    this.calories = calories;
  }

  abstract public boolean isEdible();

  public int getCalories(){
    return calories;
  }
}
```

a "promise" that every subclass implements this method

Orange.java

```
public class Orange{
  private int calories;
  private boolean isPeeled;

  public Orange(int calories){
    this.calories = calories;
    isPeeled = false;
  }
  public boolean isEdible(){
    return isPeeled;
  }
  public int getCalories(){
    return calories;
  }
  public void peel(){
    isPeeled = true;
  }
}
```

Apple.java

```
public class Apple{
  private int calories;
  private boolean isChopped;

  public Apple(int calories){
    this.calories = calories;
    isChopped = false;
  }
  public boolean isEdible(){
    return isChopped;
  }
  public int getCalories(){
    return calories;
  }
  public void chop(){
    isChopped = true;
  }
}
```

better Apple.java

```
public class Apple extends Fruit{
  private boolean isChopped;

  public Apple(int calories){
    super(calories);
    isChopped = false;
  }

  public boolean isEdible(){
    return isChopped;
  }

  public void chop(){
    isChopped = true;
  }
}
```

Fruit.java

```
public abstract class Fruit{
  private int calories;

  public Fruit(int calories){
    this.calories = calories;
  }

  abstract public boolean isEdible();

  public int getCalories(){
    return calories;
  }
}
```

better Orange.java

```
public class Orange extends Fruit{
  private boolean isPeeled;

  public Orange(int calories){
    super(calories);
    isPeeled = false;
  }

  public boolean isEdible(){
    return isPeeled;
  }

  public void peel(){
    isPeeled = true;
  }
}
```

Fruit.java

```
public abstract class Fruit{
  private int calories;

  public Fruit(int calories){
    this.calories = calories;
  }

  abstract public boolean isEdible();

  public int getCalories(){
    return calories;
  }
}
```

Abstract Class

- An abstract class allows for a *partial* implementation
  - It contains at least 1 abstract method

```
public abstract class Fruit{
   …
   abstract public boolean isEdible();
   …
}
```

Can't be instantiated, i.e.,

new Fruit(1); throws an error

Agenda

- Inheritance
  - "extends"
  - abstract class
  - interface
  - "implements"

Cookie Monster wants to eat fruits!

CookieMonster.java

```
public class CookieMonster{
  private int calories;
  public CookieMonster(){
    calories = 0;
  }
 public void eat(Cookie something){ // can only eat cookies!
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat " + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
  public static void main(String[] args){
  …
  }
}
```

4

CookieMonster.java

```
public void eat(Cookie something){
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat " + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
public void eat(Cookie something){
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat " + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
```

overloading

Fruit

Interface

- Interface simply declares methods

```
public interface Edible{
  public boolean isEdible();
  public int getCalories();
}
```

- When a class “implements” an interface, those methods are implemented in the class, e.g., Cookie implements isEdible() and getCalories()

```
public class Cookie implements Edible{
```

- An object of a class type can be stored in a variable of an interface that the class “implements”, e.g.,

```
Edible chocoCookie = new Cookie(10);
```

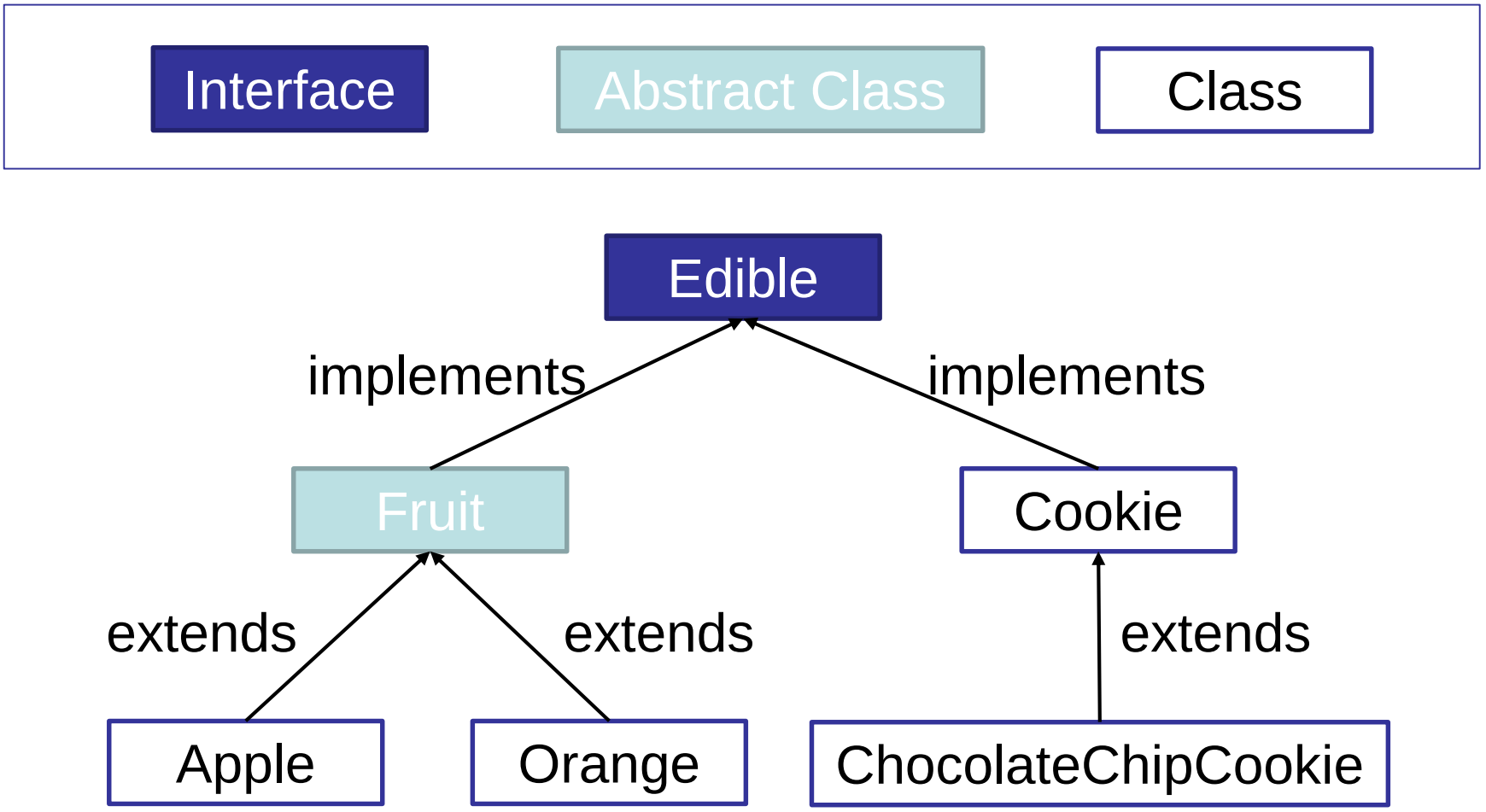
- Note, a class can **implement** many interfaces.

better CookieMonster.java

```
public void eat(Edible something){
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat " + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
```

```
public void eat(Fruit something){
    if(something.isEdible()){
      int tempCalories = something.getCalories();
      calories += tempCalories;
      System.out.println("Me eat " + tempCalories
                  +" calories! Om nom nom nom");
    }
  }
```

Example Class Hierarchy



[TAP] Dynamic Binding

```
public class Baby {
  …
  public String toString(){
     return name+ "(age "+age+")";
  }
  …
}
public class BossBaby extends Baby{
  …
  public String toString(){
     return name+ "(age "+age+", CEO)";
  }
  …
}
public class Driver{
  public static void main(String[] args){
     Baby baby = new BossBaby(3,"Bill");
     System.out.println(baby);
  }
  …
}
```

- What does the program output?
  - A. "Bill(age 3)"
  - B. "Bill(age 3, CEO)"
  - C. Memory location of baby object
  - D. Compiler error
  - E. Whatever