Help Slides
Polymorphism!

- Before reading these slides, carefully read through the handout, especially the section on support code!
- You **will** need to use polymorphism!
- It **is** your friend!
New idea: The **Game** class!

- Running a game involves a lot more logic than we’ve previously seen
  - We need to set up the components, prepare the Fruit and Bombs, decide what happens when you intersect with an object…
  - You’ll want a class to take care of this logic for you!
cs015.prj.FruitNinjaSupport.CS15Game

- We’ve provided the abstract class `CS15Game` to help with some of the game’s logic.
- Should you subclass this `CS15Game`?
  - Hint: if you look at the handout, `onBladeContact()` and `launchItem()` are abstract and don't do anything at all. How can you override them to make them do something useful?
  - `startGame()`, `addBlade(Blade blade)`, and `getChoppable()` are complete and you can use them freely.
But where is my top level class?

- In LiteBrite, we saw that conventionally, we only instantiate one object, the top level object, in the App class.
- This object is named the same name as the project.
- With FruitNinja, things are a little different:
  - You can think of your subclass of `CS15Game` as the top level class
  - Because the graphics are handled by the support code, it isn't enough to just instantiate a new subclass of `CS15Game`; we need to call the method `addGame()` from the `FruitNinjaFrame` in order to make our support code work with a class you wrote
- For the next project, you’ll be handling the graphics and we’ll return to the regular pattern of the top-level class
cs015.prj.FruitNinjaSupport.CS15Fruit

● This is an Abstract class and models a generic fruit
  ○ But, the fruit hasn’t been washed or ripened yet
  ○ Has methods to chop graphically, wash, ripen, make a splash at the fruit’s cut location, and add an image for the fruit.

● Constructor:
  ○ CS15Fruit();

● How will you subclass this CS15Fruit?
  ○ Hint: a Fruit class could model a generic fruit by extending the cs015.prj.FruitNinjaSupport.CS15Fruit class
  ○ But… we want different types of fruit! Apples, lemons, pears, oh my! Your Fruit class does not magically turn into other fruits; therefore, you should subclass Fruit even further!
Fruit Inheritance

```
cs015.prj.FruitNinjaSupport.CS15Fruit
```

Fruit

Apple  Pear  Lemon  Peach
cs015.prj.FruitNinjaSupport.CS15Bomb

- A class that models the basic capabilities of a bomb
- Methods: explode() -> graphically makes the bomb explode and ends the game.
- Just like CS15Game and CS15Fruit, this class is abstract.
Fruit vs Bomb

• What is different about Fruit and Bombs?
  o What happens to a Fruit when it is hit?
  o What happens to the game when a Bomb is hit?

• What is the same about Fruit and Bombs?
  o How do they interact with the blade?

• How can we handle different objects that share a property?
  o Maybe use a new type of class we learned about… something other classes can implement…
Choppable Interface

To ponder: what does it mean to be choppable?

Choppable

Fruit

Bomb
Launching **Fruit and Bombs**

- The next object launched should always be random – it wouldn’t be any fun if you knew exactly when you would get the next bomb!
- How can you choose a random item every time `launchItem()` is called?

**WITH A SWITCH STATEMENT!**

- Switch statements are a great way to get a new object based on a randomly chosen value
- For a randomly chosen value, use java’s built-in method `Math.random()`
Switch Statements

Rules:

- `<variable>` usually an `integer`; `char` and `enum` also possible
- `values` have to be mutually exclusive
- If `default` is not specified, Java will not do anything for unspecified values
- `break` indicates the end of a `case`—skips to end of switch statement (if you forget `break`, the code in next case will execute)

Syntax:

```java
switch (<variable>) {
    case <value>:
        // do something
        break;
    case <other value>:
        // do something else
        break;
    default:
        // take default action
}
```
Fruit vs. Bomb

Things to think about over breakfast…

● `getChoppable()` might return a Fruit or a Bomb
● What type can you use to handle both of these cases?
● Likewise, when you launch an item, you might want to launch a Fruit or a Bomb!
● Do you want to have a separate method for launching every type of Fruit, and yet another method to launch a Bomb?

Hint: Use Polymorphism!!
Containment vs. Inheritance
**Inheritance Diagram**
- Shows an inheritance hierarchy
- Shows which **subclasses** inherit from which **superclasses**
- AKA, Parent → Child Classes
- Also includes interfaces
- See above slides for example.

**Containment Diagram**
- Shows containment and associations
- **Containment**: what classes contain instances of other classes
- **Associations**: what classes a given class knows about (has a reference to)
Inheritance and containment diagrams must be SEPARATE!

Please don’t make a diagram that attempts to show both. It’s not just really confusing, it’s also incorrect. 😞
FruitNinja Containment

While there might be many Lemons/Bombs/Peaches in the game at once, the `launchItem()` handles this. You only need to return 1 of them at a time in `launchItem()`.
GOOD LUCK

ON FruitNinja!!!