

Lec 20

Inheritance, Abstract, Interfaces

WIN \$250 IN TRITON CASH



Excel with Integrity Contest

Creatively express how you excel with integrity or why you think it is important to excel with integrity!

1st Place Winner- \$250 Triton Cash

1st RunnerUp - \$150 Triton Cash

Honorable Mentions too!



Check us out for rules & details:

academicintegrity.ucsd.edu

Even more chances to win \$\$\$\$!

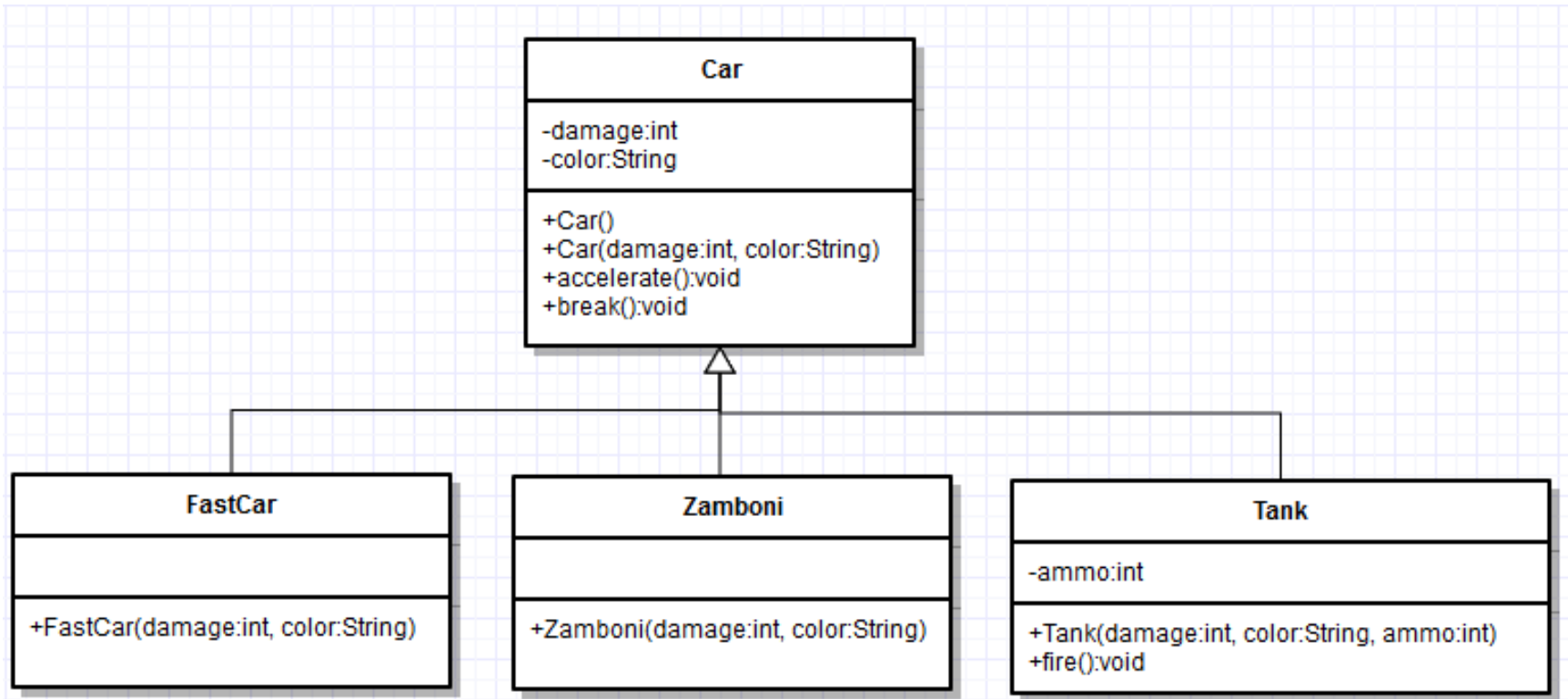
Enter the *Excel with Integrity Raffle* by following us!



@ucsdintegrity
#excelwithintegrity

POW 6 & 7

Polymorphism



Declared type vs. actual type

```
Car currentCar = new Car();
```

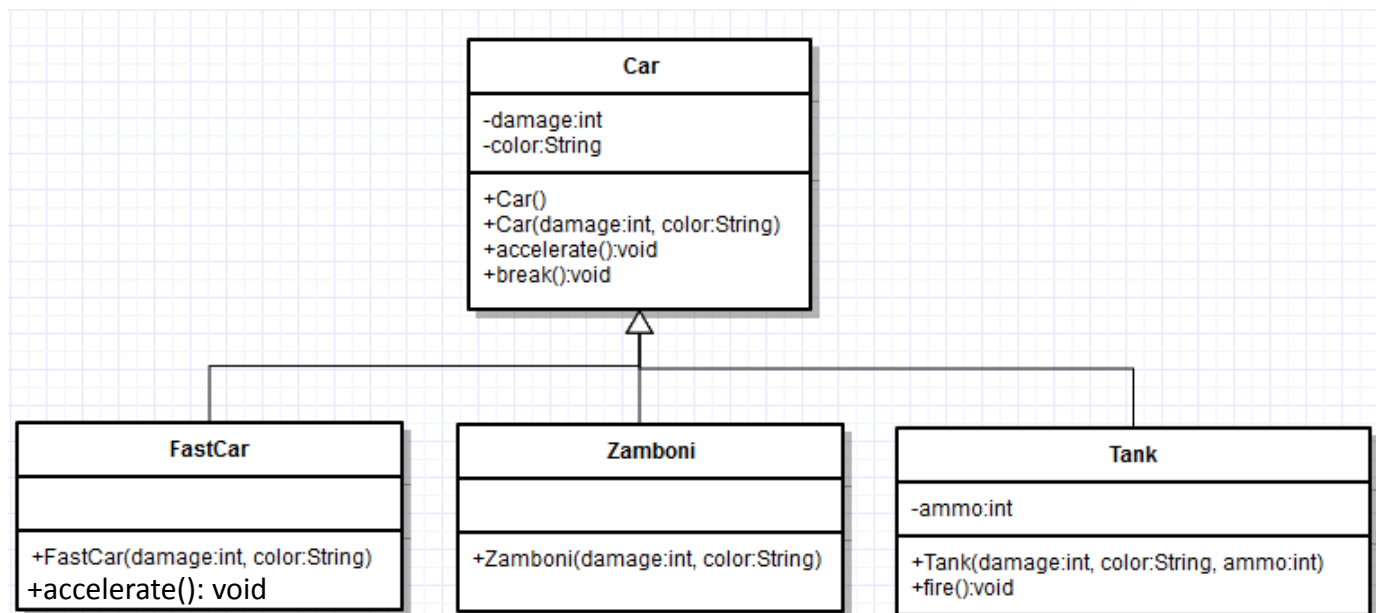
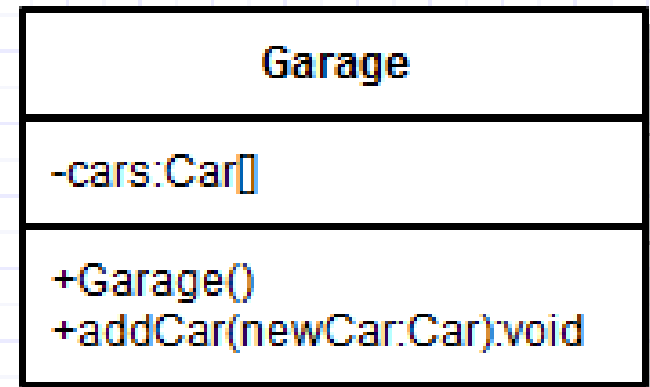
```
Car currentCar = new FastCar();
```

```
Car currentCar = new Tank();
```

```
Car currentCar = new FastCar();  
Tank purchased = new Tank();  
Garage garage = new Garage();
```

```
garage.addCar(currentCar);  
garage.addCar(purchased);
```

```
currentCar.accelerate();  
currentCar.fire(); ???
```



instanceof

<https://www.youtube.com/watch?v=XzTFWakOIVc>

```
Car currentCar = new Tank(); //we're currently driving a tank
```

```
xPressedInCar(currentCar); //somewhere in code is paying attention to  
                             what buttons on the controller are pressed
```

```
public void xPressedInCar(Car c) {  
    if(c instanceof Tank) {  
        ((Tank)c).fire();  
    }  
    else {  
        //do nothing  
    }  
}
```

- <http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>
- <http://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html>
- <http://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>
- <http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

object.toString()

```
Board b = new Board();  
ArrayList<Integer> list = new ArrayList<>();  
list.add(0);  
list.add(1);  
  
System.out.print(b);    //get Board@1234  
System.out.print(list); // get 0,1
```

println

```
public void println(Object x)
```

Prints an Object and then terminate the line. This method calls at first `String.valueOf(x)` to get the printed object's string value, then behaves as though it invokes `print(String)` and then `println()`.

Parameters:

`x` - The Object to be printed.

valueOf

```
public static String valueOf(Object obj)
```

Returns the string representation of the Object argument.

Parameters:

`obj` - an Object.

Returns:

if the argument is `null`, then a string equal to "null"; otherwise, the value of `obj.toString()` is returned.

See Also:

`Object.toString()`

toString

```
public String toString()
```

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the a other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

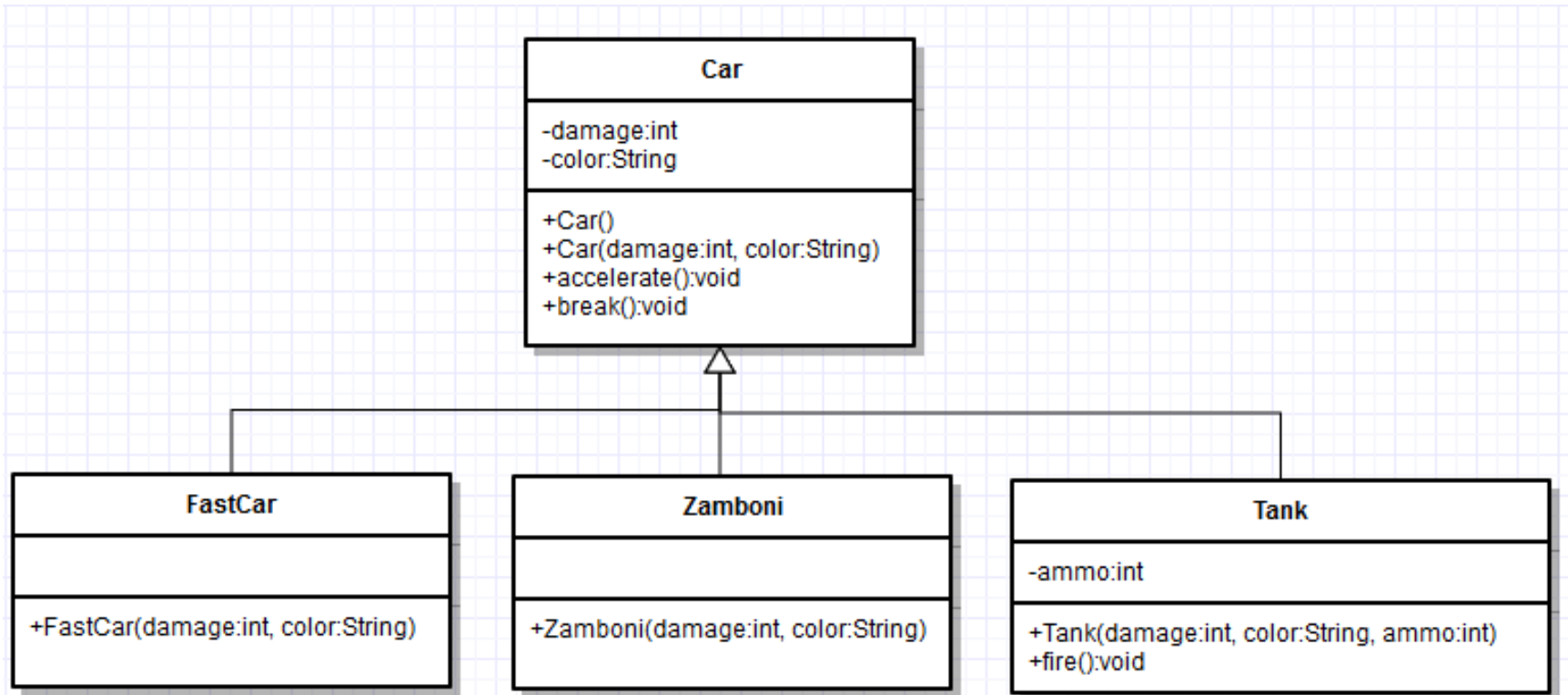
Returns:

a string representation of the object.

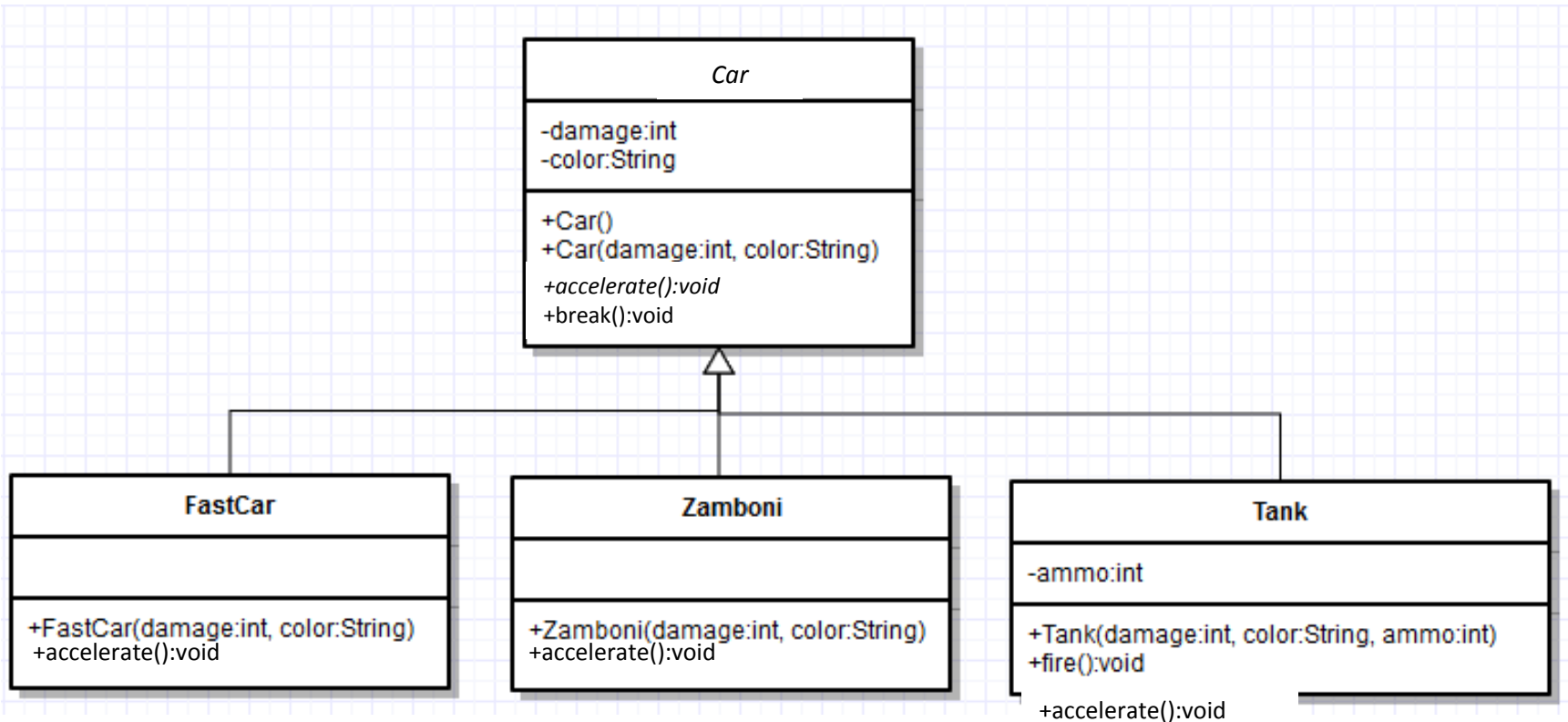
Overriding methods in HW8

Abstract classes and Interfaces

Abstract Classes



Abstract Classes



```
public abstract class Fruit{  
    private String expirationDate;  
    public abstract void peel();  
    ... // getExpDate(), ctors, etc.  
}
```

Two blue arrows point upwards from the Apple and Orange class boxes to the Fruit class box, indicating that both Apple and Orange inherit from Fruit.

```
public class Apple extends Fruit{  
    ...  
    @Override  
    public void peel() {  
        System.out.println("no need");  
    }  
    ...  
    // Things that are common to apples  
}
```

```
public class Orange extends Fruit{  
    ...  
    @Override  
    public void peel() {  
        System.out.println("remove rind");  
    }  
    ...  
    // Things that are common to oranges  
}
```

```
Fruit[] bought;  
// populate array with various types of fruit (apple/orange)  
  
for ( Fruit f : bought)  
    System.out.println( f.getExpDate() + " peel: " + f.peel() );
```

Abstract Classes – Key Points

Abstract Classes

Declared with keyword **abstract**

A class with at least 1 abstract method must be defined abstract

Use keyword **extends** to inherit from abstract class

Cannot create an instance of an abstract class:



```
Fruit f = new Fruit();
```

Interfaces

```
public interface Food {  
    public void eat();  
}
```



```
public class Apple implements Food {  
    ...  
    @Override  
    public void eat() {  
        System.out.println("eat more");  
    }  
}
```

```
public class Horse implements Food {  
    ...  
    @Override  
    public void eat() {  
        System.out.println("too hungry");  
    }  
    ...  
}
```

```
Food[] toEat;  
// populate array with various types of food  
  
for ( Food f : toEat)  
    System.out.println( f.eat() );
```

Interfaces – Key Points

Interfaces

Used in place of an abstract class when there is no default implementation to inherit

Define with keyword **interface** (vs. keyword **class**)

Can only declare **public** method declarations and **public static final** constants

Interfaces – Key Points

Interfaces

Use keyword **implements** to inherit interface

Tells compiler this class intends to implement (provide full method bodies) all the abstract method signatures from the interface

Key for polymorphism

A class can implement multiple interfaces

What gets printed

```
public interface ActionListener {  
    public void actionPerformed();  
}
```

```
public class Button1 implements ActionListener  
{  
    public void saveFile() {  
        System.out.println("file saved");  
    }  
    public void actionPerformed() {  
        saveFile();  
    }  
}
```

```
public class Label1 implements ActionListener {  
    public void updateText () {  
        System.out.println("closing");  
    }  
    public void actionPerformed() {  
        updateText();  
    }  
}
```

```
ActionListener button = new Button1();  
ActionListener label = new Label1();  
  
button.actionPerformed();  
label.actionPerformed();
```

- A) Compiler error
- B) closing, file saved
- C) file saved, closing
- D) Nothing
- E) Run time error

A) Yes

B) No

Suppose A is an interface.

Can you create an instance using `new A()`

Can you declare a reference var x as: `A x;`

Suppose A is an abstract class.

Can you create an instance using `new A()`

Can you declare a reference var x as: `A x;`