Code Reuse Through Hierarchies

Part 2: Within an inheritance hierarchy you will learn: the effect type and type conversion, how to declare the type for a container of parent and child classes and how class `Object` is the parent of all Java classes.

James Tam

Review: Casting

- The casting operator can be used to convert between types.
- **Format**:

```
<Variable name> = (type to convert to) <Variable name>;
```

- **Example (casting needed: going from more to less)**

```
double full_amount = 1.9;
int dollars = (int) full_amount;
```

- **Example (casting not needed: going from less to more)**

```
int dollars = 2;
double full_amount = dollars;
```

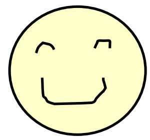
James Tam

Real Life Examples: Expectations Vs. Reality

Getting more than expected: acceptable

You are owed $100

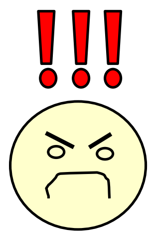
You receive $1000



Getting less than expected: not acceptable

You are owed $100

You receive $10

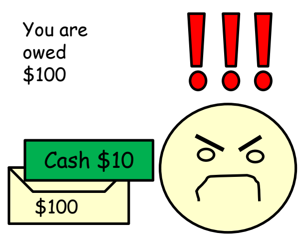


James Tam

Real Life Examples: Expectations Vs. Reality (2)

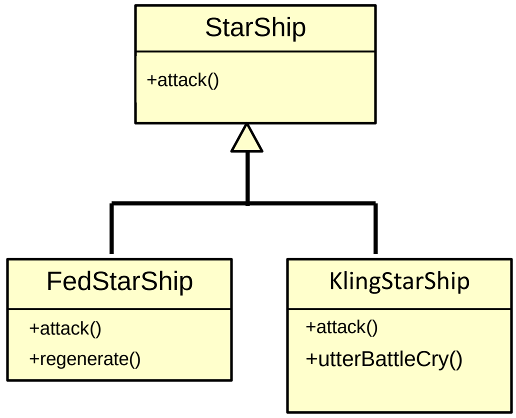
Misrepresenting reality: still not acceptable in the end

You are owed $100

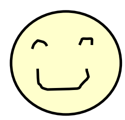


James Tam

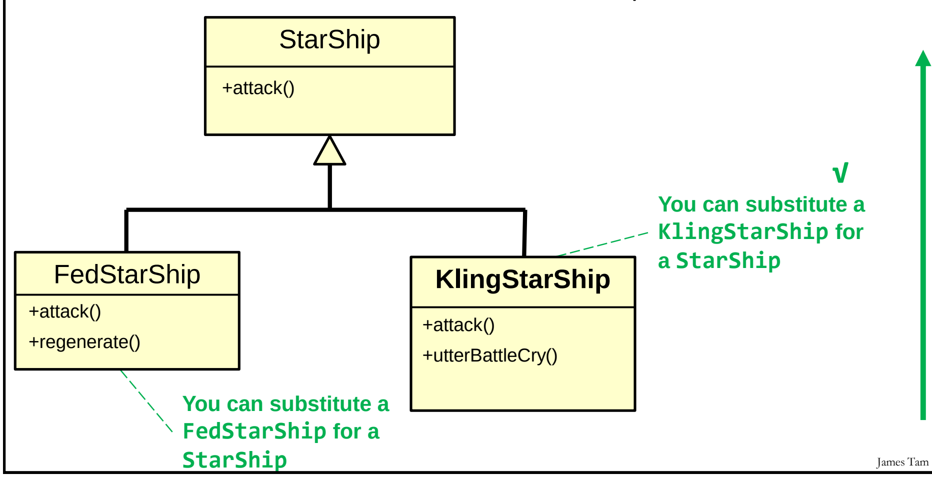
Example Inheritance Hierarchy



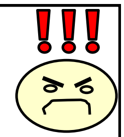
Casting And Inheritance (Up)



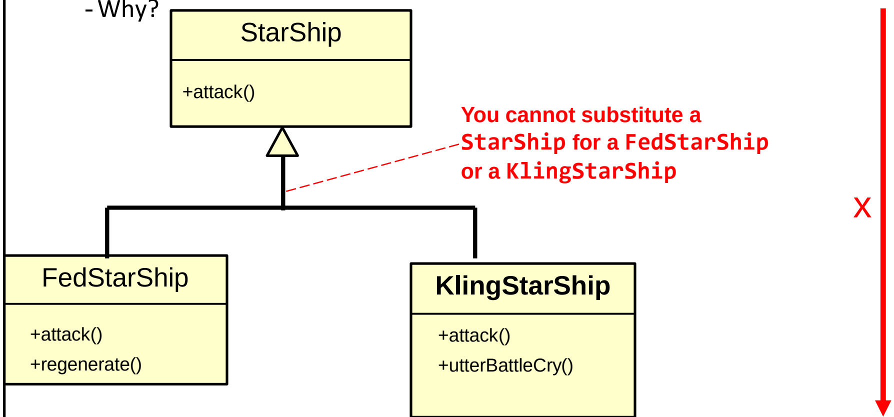
- Because the child class IS-A parent class you can substitute instances of a subclass for instances of a superclass.



Casting And Inheritance (Down)



- You cannot substitute instances of a superclass for instances of a subclass
  - Why?



James Tam

Reminder: Operations Depends On Type

- Sometimes the same symbol performs different operations depending upon the type of the operands/inputs.
- Example:

```
int num1 = 2;
int num2 = 3;
num1 = num1 + num2;
```

Vs.

```
String aString = "foo" + "bar";
```

- Some operations won’t work on some types
- Example:

```
String aString = 2 / 3;
```

James Tam

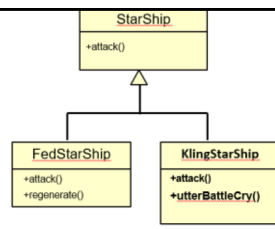
Reminder: Behavior Depends Upon Class Type

- The methods that can be invoked by an object depend on the class definition
- Example:

```
class X                         class Y
{                               {
    method1() {                      method2() {
    }                                 }
}                               }

X x1 = new X();
x1.method1();      //Yes
Y y1= new Y();
y1.method1();      //No
```

James Tam

Casting And Inheritance




```
  StarShip regular = new StarShip();
  KlingStarShip kling = new KlingStarShip();

X regular.utterBattleCry();  //Won't compile: no such method.

  regular = kling;
  //Won't compile: I think I point to the wrong type
X regular.utterBattleCry();

  //Works - this time but a dangerous cast
  ((KlingStarShip) regular).utterBattleCry();

  regular = new StarShip();
  kling = (KlingStarShip) regular; //Dangerous cast crashes it.
X kling.utterBattleCry();  //Inappropriate action for type
```

James Tam

Caution About Class Casting: Check First!

- When casting between classes only use the casting operator if you are sure of the type!
- Check if an object is of a particular type is via the `instanceof` operator
- (When used in an expression the `instanceof` operator returns a boolean result)
- Format:

```
if (<reference name> instanceof <class name>)
```

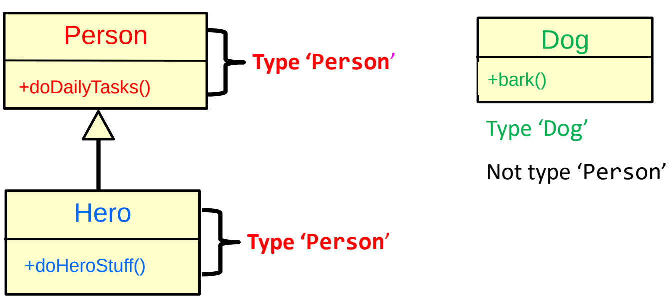
- Example:

```
if (supPerson instanceof Person)
```

James Tam

Instanceof Example

- **Name of the folder containing the full online example**: `6typeCheck`



James Tam

Driver.main()

```
Person regPerson = new Person();
Hero supPerson = new Hero();
Dog rover = new Dog();

//Instanceof checks if the object is a certain type or
//a subclass of that type (e.g., a Hero is a Person)
if (regPerson instanceof Person)
    System.out.println("regPerson is a type of Person");
if (supPerson instanceof Person)
    System.out.println("supPerson is also a type of Person");

//Checks for non-hierarchical: Compiler prevents nonsensical
//checks
//if (rover instanceof Person)
//    System.out.println("Rover is also a type of Person");
```

```
regPerson is a type of Pers
```

```
supPerson is also a type of Person
```

James Tam

Driver.main(): 2

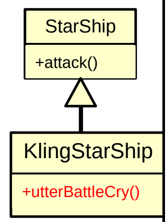
```
if (supPerson instanceof Hero)
    System.out.println("supPerson is a type of Hero");

//Checks within hierarchy: Compiler doesn't prevent
if (regPerson instanceof Hero)
    System.out.println("[Should never appear]: regPerson is a
                        type of Hero");
```

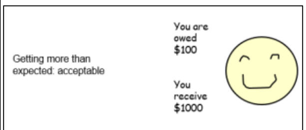
```
supPerson is a type of Hero
```

James Tam

Containers: Homogeneous

- Recall that arrays must be homogeneous: all elements must be of the same type e.g., `int [] grades`
- **Again recall**: A child class is an instance of the parent (a more specific instance with more capabilities).





- If a container, such as an array is needed for use in conjunction with an inheritance hierarchy then the type of each element can simply be the parent.

```
StarShip [] array = new StarShip[2];
array[0] = new StarShip();  // [0] wants a StarShip, gets a StarShip
array[1] = new KlingStarShip();  // [1] wants a StarShip, gets a
                                //  KlingStarShip (even better!)
```

James Tam

The Parent Of All Classes

- You've already employed inheritance.
- Class `Object` is at the top of the inheritance hierarchy.
- Inheritance from class `Object` is implicit.
- All other classes automatically inherit its attributes and methods (left and right are logically the same)

```
class Person                  class Person extends Object
{                             {

}                             }
```

  - e.g., "`toString()`" are available to its child classes
- For more information about this class see the url (last visited 2021):
  - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

James Tam

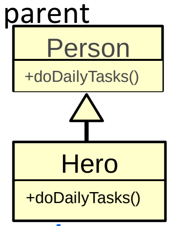
The Parent Of All Classes (2)

- This means that if you want to have an array that can contain *any type* of Java object then it can be an array of type `Object`.
  - Object [] array = new Object[SIZE];
- Built-in array-like classes such as class `Vector` (an array that 'automatically' resizes itself consists of an array attribute whose type is class `Object`)
  - For more information on class `Vector` (last visited 2021):
  - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Vector.html

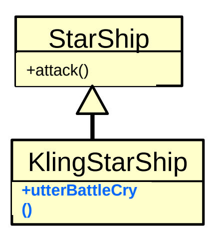
James Tam

Determining Type: Hierarchies

- As mentioned: normally **type checking should not be needed for a polymorphic method** (the child class overrides a parent method).
  - No `instanceof` needed



- However type checking is **needed if a method specific to the child** is being invoked.
  - Check the type with the `instanceof` is needed



James Tam

Example: Containers With 'Different' Types

- **Name of the folder containing the complete example**: `7hierarchiesContainment`

James Tam

Class `StarShip`

```
public class StarShip {
    public static final int MAX_HULL = 400;
    public static final char DEFAULT_APPEARANCE = 'C';
    public static final int MAX_DAMAGE = 50;
    private char appearance;
    private int hullValue;

    public StarShip() {
        appearance = DEFAULT_APPEARANCE;
        hullValue = MAX_HULL;
    }

    public StarShip (int hull) {
        appearance = DEFAULT_APPEARANCE;
        hullValue = hull;
    }
```

James Tam

Class StarShip (2)

```
    public StarShip (char newAppearance) {
        this();
        appearance = newAppearance;
    }

    public int attack() {
        System.out.println("<<< StarShip.attack() >>>");
        return(MAX_DAMAGE);
    }
```

James Tam

Class StarShip (3): Get()s, Set()s

```
    public char getAppearance() {
        return appearance;
    }

    public int getHullValue() {
        return(hullValue);
    }

    public void setAppearance(char newAppearance) {
        appearance = newAppearance;
    }

    public void setHull(int newHullValue) {
        hullValue = newHullValue;
    }
}
```

James Tam

Class FedStarShip

```
public class FedStarShip extends StarShip {
    public static final int MAX_HULL = 800;
    public static final char DEFAULT_APPEARANCE = 'F';
    public static final int MAX_DIE_ROLL = 6;
    public static final int DIE_ROLL_BOOSTER = 1;
    public static final int NUM_DICE = 20;

    public FedStarShip() {
        super();
        setHull(MAX_HULL);    //800 not 400 due to shadowing
        setAppearance(DEFAULT_APPEARANCE);  //'F' not 'C'
    }

    public void regenerate() {  //Unique method
        int temp = hullValue + 40;
        if (temp <= MAX_HULL)
            hullValue = temp;
    }
```

Shadows parent constants

James Tam

Class FedStarShip (2)

```
     //Overridden / polymorphic method
     public int attack() {
        System.out.println("<<< FedStarShip.attack() >>>");
        Random aGenerator = new Random();
        int i = 0;
        int tempDamage = 0;
        int totalDamage = 0;
                                  20
        for (i = 0; i < NUM_DICE; i++)
        {
                                                     6
            tempDamage = aGenerator.nextInt(MAX_DIE_ROLL) +
                        1DIE_ROLL_BOOSTER;
            totalDamage = totalDamage + tempDamage;
        }
        return(totalDamage);
    }
}
```

James Tam

Class KlingStarShip

```
public class KlingStarShip extends StarShip {
    public static final char DEFAULT_APPEARANCE = 'K';
    public static final int MAX_DIE_ROLL = 12;
    public static final int DIE_ROLL_BOOSTER = 1;
    public static final int NUM_DICE = 20;

    public KlingStarShip() {
        super();
        setAppearance(DEFAULT_APPEARANCE);
    }
    //Unique to KlingStarShip objects
    public void utterBattleCry() {
        System.out.println("Heghlu'meH QaQ jajvam!");
    }
}
```

James Tam

Class KlingStarShip (2)

```
   //Overridden / polymorphic method
   public int attack() {
        System.out.println("<<< KlingStarShip.attack() >>>");
        Random aGenerator = new Random();
        int i = 0;
        int tempDamage = 0;
        int totalDamage = 0;

                          20
        for (i = 0; i < NUM_DICE; i++) {
                                                      12
            tempDamage = aGenerator.nextInt(MAX_DIE_ROLL) +
                        1DIE_ROLL_BOOSTER;
            totalDamage = totalDamage + tempDamage;
        }
        return(totalDamage);
    }
```

James Tam

Driver Class: SpaceSimulator

```
public class SpaceSimulator
{
    public static void main(String [] args)
    {
        Galaxy alpha = new Galaxy();
        alpha.display();
        alpha.runSimulatedAttacks();
    }
}
```

James Tam

Class Galaxy

```
public class Galaxy {
    public static final int SIZE = 4;
    private StarShip [][] grid;
```

James Tam

Class Galaxy (2)

```
public Galaxy() {
    boolean squareOccupied = false;
    grid = new StarShip [SIZE][SIZE];
    int r;
    int c;
    int hull;

    for (r = 0; r < SIZE; r++) {
        for (c = 0; c < SIZE; c++)
        {
            grid[r][c] = null;
        }
    }
    grid[0][0] = new FedStarShip();
    grid[0][1] = new KlingStarShip();
    grid[1][0] = new StarShip();
}
```

```
FK
C
```

James Tam

Class Galaxy (3)

```
public void runSimulatedAttacks() {
    int damage;
    damage = grid[0][0].attack();
    System.out.println("Fed ship attacks for: " + damage);
    System.out.println();
```

```
<<< FedStarShip.attack() >>>
Fed ship attacks for: 66
```

```
    damage = grid[0][1].attack();
    System.out.println("Kling ship attacks for: " + damage);
    System.out.println();
```

```
<<< KlingStarShip.attack() >>>
Kling ship attacks for: 116
```

```
    damage = grid[1][0].attack();
    System.out.println("
      damage);
    System.out.println();
```

```
<<< StarShip.attack() >>>
Old style ship attacks for: 50
```

Type check not needed because:

`attack()` is overridden / polymorphic

James Tam

Class Galaxy (4)

```
FK
C
```

```
/* Won't work because it's an array of references
   to  StarShips not KlingStarShips.
  grid[1][0].utterBattleCry(); */
```

Type check 'instanceof' needed because:

Array of StarShips but utterBattleCry() unique to KlingStarShip

```
if (grid[0][0] instanceof KlingStarShip)
    ((KlingStarShip) grid[0][0]).utterBattleCry();
```

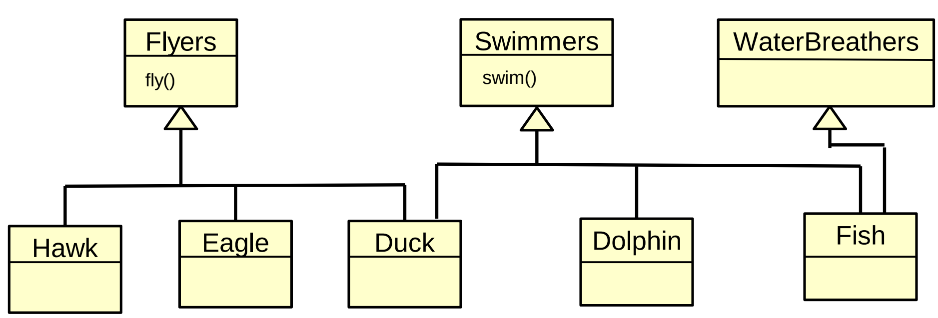
```
Heghlu'meH QaQ jajvam!
```

```
if (grid[0][1] instanceof KlingStarShip)
    ((KlingStarShip) grid[0][1]).utterBattleCry();

if (grid[1][0] instanceof KlingStarShip)
    ((KlingStarShip) grid[1][0]).utterBattleCry();

    }
} // End runSimulatedAttacks()
```

James Tam

Multiple Inheritance

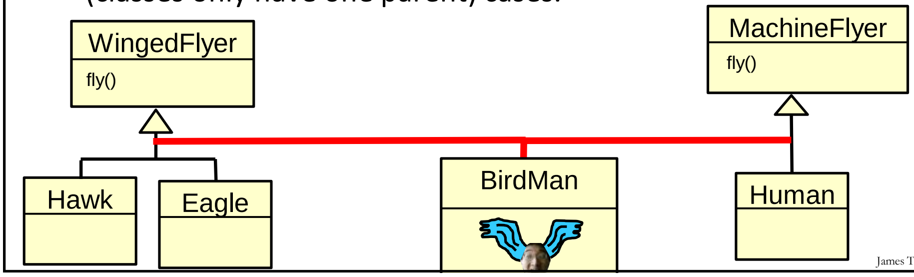
- What happens if some behaviors or attributes are common to a group of classes but some of those classes include behaviors shared with other groups?
- Or some groups of classes share some behaviors but not others?



James Tam

Multiple Inheritance (2)

- It is implemented in some languages e.g., C++
- It is not implemented in other languages e.g., Java
- Pro: It allows for more than one parent class
  - (JT: rarely needed but nice to have that capability for that odd exceptional case).
- Con: Languages that allow for multiple inheritance require a more complex implementation even for single inheritance (classes only have one parent) cases.



James Tam

You Should Now Know

- How to call methods that are unique to a child class when the type may be either parent or a child.
- How casting works within an inheritance hierarchy
  - When the `instanceof` operator should and should not be used to check for type in an inheritance hierarchy
- Class `Object` is the parent of all classes in Java
  - Capabilities inherited from the parent (if you refer to the API for class Object)
- How homogeneous composite types (such as arrays) can appear to contain multiple types within one container

James Tam

Copyright Notification