

Introduction to Programming

with Java, for Beginners

Inheritance

Example: Bot and BetterBot

```
public class Bot{
    private int x;
    private int y;

    public int getX() { .. }
    public int getY() { .. }

    public void eatDot(){ ..}
    public void move(){.. }
    public void turnLeft(){.. }
}

public class BetterBot extends Bot{

    public void turnRight(){
        turnLeft();
        turnLeft();
        turnLeft();
    }
}
```

ESE112

1/11

Inheritance

One of key concepts of OOP

- A hierarchical relationship among classes
- Establishes a superclass/subclass relationship
- Establishes "is a" relationships
 - e.g. a BetterBot "is a" Bot

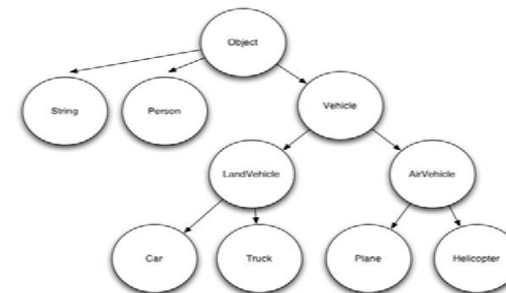
Benefits:

- Reusability of code
 - Put code in one class, use it in all the subclasses
 - Revisions only needs to be done in 1 place
- Polymorphic code (works on "many forms")
 - Write general purpose code designed for a supertype that works for all subtypes
- Polymorphic data structures
 - E.g. an array of a supertype can hold objects of any subtype

ESE112

2/11

Inheritance Tree



- Java has *single inheritance*; each node has one parent
- (Except for Object which has no parent)
- C++ has *multiple inheritance*, but it's messy

ESE112

3/11

The Object Class

All classes inherit from the Object class

- The Object class is the root of the class hierarchy
- When we create a new class, “extends Object” is implied/implicit

```
public class Car {  
}  
  
public class Car extends Object{  
}
```

The Object class has several methods which all object *inherit*, most notably:

toString() and equals()

ESE112

4/11

The toString() Method

By default, it returns a String containing an object’s heap address

By convention, it is *overridden* to describe the object’s state

Most common usage: debugging

```
public class Car {  
    private int miles;  
    private String model;  
}  
  
public Car(String model) {  
    this.model = model;  
    miles = 0;  
}  
  
public String toString() {  
    return "model: " + model + ", miles: " + miles;  
}
```

Recall toString()
from RealVector
Assignment

ESE112

5/11

The equals() Method

By default, compares heap addresses

By convention, it is *overridden* to match the developer’s notion of equality

```
public class Person {  
    private int social; // social security #  
    private String name;  
}  
  
public int getSocial() { return social; }  
  
public boolean equals(Person p) {  
    return social == p.getSocial();  
}
```

ESE112

6/11

The “extends” keyword

Inheritance is established via the “extends” keyword

```
public class Bot{  
}  
  
public class BetterBot extends Bot{  
}
```

Now we say

- BetterBot *inherits* from Bot
(it gets all the instance variables and methods defined in Bot unless they are declared to be “final”)
- A BetterBot “is a” Bot
- BetterBot is a *subclass/subtype* of Bot.
- Bot is the *superclass/supertype* of BetterBot

ESE112

7/11

Type Rules

A reference variable of type *t* may hold a value of its own type or any subtype (but not of a supertype).

Given the following variable declaration:

```
Bot b;
```

Which of the following assignments are valid?

```
b = new Bot();
b = new String();
b = new BetterBot();
b = new Object();
```

How about these?

```
BetterBot bb;
bb = new Bot();
bb = new String();
bb = new BetterBot();
bb = new Object();
```

ESE112

8/11

The “instanceof” Operator

```
> Bot b = new Bot();
> b instanceof Bot
true
> b instanceof Object
true
> b instanceof BetterBot
false
> BetterBot bb = new BetterBot();
> bb instanceof BetterBot
true
> bb instanceof Object
true
> bb instanceof Bot
true
> Bot bbb = new BetterBot(); //a variable can store a subtype
> bbb instanceof BetterBot()
true
bbb instanceof Bot
true
```

ESE112

9/11

Another Example

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

```
    Dog(String dogName, int dogAge){
        name = dogName;
        age = dogAge;
    }
}
```

```
class BetterDog extends Dog{
    ....
}
```

ESE112

10/11

Constructors and Inheritance

```
> BetterDog d = new BetterDog();
```

When an object is created, its constructor is called

- But first, a constructor from its highest ancestor (Object) is called, then the next highest (Dog), then its own (BetterDog)
- The default behavior is to that the default (no-argument constructor) is executed

A constructor can explicitly call its parent's (its superclass') constructor by making a call to `super()`

```
class BetterDog extends Dog{

    public BetterDog(String name, int age){
        super(name, age);
    }
    ...
}
```

ESE112

11/11