TINKER ACADEMY

Programming Using Java

Handout 6: Inheritance, Packages and Interfaces Part 2

Note your Student ID. You will need to use it throughout the Course.

Setup Instructions In Classroom

Connect to the Local Class Network

1. Select WiFi “TINKER ACADEMY”
2. This network has only LOCAL access and does NOT connect to the internet

Update the Course

1. Ensure you are connected to “TINKER ACADEMY”
2. Restart the VM. Login into the VM.
3. Open Firefox in the VM
4. Your Instructor would tell you what to type in the browser. (Typically it is 192.168.1.5)
5. You should see a page with a list of entries.
6. Click on CourseUpdate<Date>.zip. This will download CourseUpdate<Date>.zip onto your VM
7. Open Nautilus. Click on Downloads. You should see the file CourseUpdate<Date>.zip
8. Right Click on CourseUpdate<Date>.zip. Select Extract Here.
9. Open the extracted folder
10. Double click Course Update. Select “Run” in the window.

Update the Course (Alternate Approach In Class Using USB)

1. Borrow a USB drive from the Instructor
2. If you are on VirtualBox
    a. Click on Devices in the Top level Menu
    b. Select Drag ‘n’ Drop
    c. Select Bidirectional
3. If you are on VirtualBox (Another Way)
    a. Shutdown Virtual Machine
    b. Click on VM in the VirtualBox Manager
    c. Click on the Settings
    d. Click General
    e. Click Advanced Tab

© Tinker Academy 2014

    f. Select “Bidirectional” under Drag ‘n’ Drop
    g. Click OK
    h. Start Virtual Machine
4. If you are on VMWare
    a. Open the virtual machine settings editor (VM > Settings),
    b. Click the Options tab
    c. Select Guest isolation.
    d. Deselect Disable drag and drop to and from this virtual machine
5. Open Nautilus, Click on Desktop
6. Drag the file CourseUpdate<Date>.zip from Windows or Mac onto Desktop in your Virtual Machine
7. Right Click on CourseUpdate<Date>.zip. Select Extract Here.
8. Open the extracted folder
9. Double click Course Update. Select “Run” in the window.
10. Eject the USB Drive and hand it back to the Tinker Academy instructor

© Tinker Academy 2014

Setup Instructions At Home

Connect to your Home WiFi Network

Updating the Course (Using Wifi)

1. Make sure you are on the Home WiFi Network.
2. Click the “Setup” folder in “Nautilus” under “Bookmarks”
3. Double click “Course Update”. Choose “Run”.
   You should see a message “Update course in progress”
   If you see a window popup with the message “update course failed”.
   Hop onto Skype, and request help in the class chat group.
   And send an email to classes@tinkeracademy.com with your name and student ID.
4. Wait for a few minutes (to allow for the update to complete)
5. Follow the instructions in this handout (last 2 pages) on the quiz and homework steps.

Submitting Quiz and Homework

1. Make sure you are on the Home WiFi Network.
2. Click the “Setup” folder in “Nautilus” under “Bookmarks”
3. Double click “Course Submit”. Choose “Run”.
   If you see a window popup with the message “submit course failed”.
   Hop onto Skype, and request help in the class chat group.
   And send an email to classes@tinkeracademy.com with your name and student ID.

© Tinker Academy 2014

Virtual Machine Installation

Installing the Virtual Machine (VM)

1. Borrow the USB drive from your Tinker Academy instructor
2. Create the folder “tinkeracademy” (without the quotes) under Documents using Finder or Windows Explorer. Type it in *exactly* as indicated.
3. Copy the folder “installers” from the USB drive to under “tinkeracademy” using Finder or Windows Explorer
4. Eject the USB Drive and hand it back to the Tinker Academy instructor
5. Locate the VirtualBox installer under “tinkeracademy” using Finder or Windows Explorer

If your Laptop is	Double click on
Windows 7	VirtualBox-4.3.12-93733-Win.exe
Windows 8	VirtualBox-4.3.14-95030-Win.exe
Mac	VirtualBox-4.2.26-95022-OSX.dmg

6. Install the VirtualBox application
7. Congratulations, You completed a major milestone. Give yourself a pat on the back :)

Importing the Virtual Machine (VM)

1. Locate the Virtual Machine “tinkeracademy.ova” under “tinkeracademy”
2. Double click on “tinkeracademy.ova”. You should get the import screen in VirtualBox with an “Import” Button. Click on the “Import” button to Import the Virtual Machine.

Starting the Virtual Machine (VM)

1. Once the Import is complete and successful, you should see the VM “TinkerAcademy” in the side panel in VirtualBox.
2. If it says “Powered Off” click on the Start Button (Green Arrow) in the VirtualBox Toolbar. This will start the VM.
3. If it says “Running” click on the Show Button (Green Arrow) in the VirtualBox Toolbar. This should display the VM window.
4. Once the VM starts up you will be presented with a login screen. Type in “password” without the quotes. Type it in exactly as indicated and hit “Enter”.
5. Once the login is completed you should see a Desktop with a few icons. The Screen might go fuzzy for a few seconds before displaying the Desktop. *That is ok.*
6. Congratulations. You are now running Linux within your laptop.
7. Double click on the “Firefox” icon in the Sidebar. This should launch Firefox. Verify you have network access. Close “Firefox”

© Tinker Academy 2014

Launching the Virtual Machine in Full Screen

1. Use the VirtualBox menu View->Switch to Fullscreen to switch the VM to fullscreen mode
2. Use the same VirtualBox menu View->Switch to Fullscreen to switch the VM back out of fullscreen mode

Shutting Down the Virtual Machine

1. Click on the red close window button (to the top left on a Mac, top right in Windows).
2. You will prompted with a confirmation message asking if you want to “Power Off” the machine. Click the button to confirm power off.
3. In a few minutes the VM will shut down and you should see the VirtualBox side panel with the “Tinker academy” VM indicating “Powered Off”.

Restarting the Virtual Machine

1. Start VirtualBox
2. Click on the VM “TinkerAcademy” in the VirtualBox side panel.
3. Click on the Start Button (Green Arrow) in the VirtualBox Toolbar. This will start the VM.
4. Once the VM startup you will be presented with a login screen.

Right Click in VM on Mac

1. Open System Preferences, Trackpad
2. Enable “Secondary Click”, Toggle the small arrow to the right and select “Click with two fingers”.

© Tinker Academy 2014

Getting Ready to Program

Import StarterPack6.zip
We will be using StarterPack6.zip for this class.

Click on StarterPack6.zip under “Courses”. Right Click. Select “Extract Here”
Start Eclipse from Desktop. Right Click over Package explorer. Select “Import”. Browse and select the extracted folder “StarterPack”. Click “Finish” to complete the Import.

Structure of StarterPack6.zip
StarterPack6 is an Eclipse Java Project. The Project has

1. A Java Source File called “StarterPack6.java”
2. A Java Source File called “MyMinecraftHouse.java”
3. A Java Source File called “MyMinecraftWoodHouse.java”
4. A Java Source File called “MyMysteryWoodHouseGenerator.java”
5. Project references to the Java Runtime Environment 1.6 System Library.
6. Project references to the Minecraft Plugin Library.
7. The Bukkit configuration file called plugin.yml

Run the Program

1. Click over project name “StarterPack6”.
2. Toggle the src folder to select StarterPack6.java
3. Right Click. Select “Run As…”. Select “Java Application”.
   The text “Starter Pack 6” should be displayed in the Console window (bottom)

If the “Console” window is not visible click on the “Window” Toolbar (top), select “Show View”, select “Console”. If all else fails, click on the “Window” Toolbar (top), select “Reset Perspective…” click “OK”. This will put Eclipse back into the default Factory state. You can now use “Show View” to view the “Console”.

© Tinker Academy 2014

Refresher

Java Inheritance

What is **inheritance**?
A class is a blueprint that is used to create **objects** of that type.

The blueprint defines the state and behavior of objects of that type.

Inheritance is a **language mechanism** which allows one class to base its state and behavior off another class.

The class that inherits from another class is called the "child" class or the "sub" class or the "derived" class.

The class that is being inherited from is called the "parent" class or the "super" class or the "base" class.

The inheritance used in Java is called class based inheritance.

Visual Model
The visual model below below shows the blueprint for MyMinecraftWoodHouse, MyMinecraftHouse.

The classes MyMinecraftWoodHouse and MyMinecraftHouse were defined in Handout 5.

The object myHouse is based on the blueprint MyMinecraftWoodHouse and has the datatype MyMinecraftWoodHouse.

Class	Method 0	Method 1	Method 2	Method 3
MyMinecraftWoodHouse	Constructor	build		
Superclass	Field 1	Field 2	Field 3	
MyMinecraftHouse				

Class	Method 0	Method 1	Method 2	Method 3

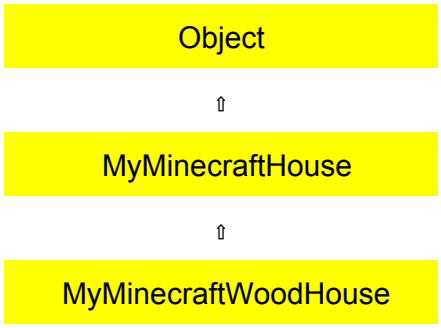
© Tinker Academy 2014

MyMinecraftHouse	Constructor	build		
	Field 1	Field 2	Field 3	
	name	width	height	

Object	Blueprint 1	name	width	height
myHouse	MyMinecraftWoodHouse	Casa Rosa	10	10

Another Visual Model

Unified Modeling Language Class Diagram
The up arrow points from the subclass to the super class



Special Privileges of a Subclass

Privilege #1 Access to the Superclass Constructors
The constructors of the subclass get to access the constructors of the superclass using the super keyword.

Privilege #2 Inheritance of Fields and Methods

The subclass inherits all fields that are declared in the superclass, if the field has a protected access modifier.

Likewise, the subclass inherits all methods that are declared in the superclass, if the method has a protected access modifier.

Privilege #3 Assignment to variables of Superclass datatype
Objects that are created using the new operator from subclasses can be assigned to a variable of the superclass datatype.

© Tinker Academy 2014

Method Overriding

A subclass can declare a method with the same name as in the superclass. If the method signature and return type is the same as in the superclass, the method declaration in the subclass is said to override the method declaration in the superclass.

Abstract Modifiers

The abstract class modifier indicates that the class is incomplete or should be considered incomplete

The abstract method modifier indicates that the method is declared in the class but not implemented in the class. Any subclass that is not abstract needs to implement the method.

Refresher Questions

Question	Your Answer
The super class we created in Handout5 is _____	
The sub class we created in Handout5 is _____	
The abstract class modifier indicates that _____	
The abstract method modifier indicates that _____	
Does **every** class have a super class?	
The ancestor of all classes is the _____ class	
The constructors of the subclass get to access the constructors of the superclass using the _____ keyword.	
Subclasses inherit the field declared in the superclass, if the field has the _____ access modifier.	
Subclasses inherit the method declared in the superclass, if the method has the _____ access modifier	

© Tinker Academy 2014

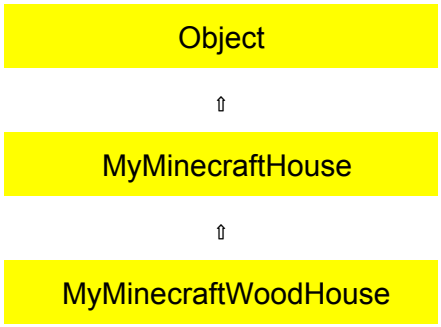
Is this valid? MyMinecraftHouse myHouse = new MyMinecraftWoodHouse(“Casa Rosa”,10,10)	

© Tinker Academy 2014

Class Hierarchy

Unified Modeling Language (UML) Class Diagram

The up arrow points from the subclass to the super class



The UML Class Diagram shows that

- MyMinecraftHouse is the superclass of MyMinecraftWoodHouse and
- Object is the superclass of MyMinecraftHouse.

The classes form a hierarchy. Such a hierarchy is called the class hierarchy of the class MyMinecraftWoodHouse.

The hierarchy starts at MyMinecraftWoodHouse and ends at Object. MyMinecraftWoodHouse, MyMinecaftHouse and Object belong to the hierarchy.

The class hierarchy always ends at the **root** of all classes, the Object class.

The class hierarchy is a visual way to easily see which object can be assigned to a variable.

Consider the following variable assignments

Variable Assignment	Allowed?
Object myObject = new MyMinecraftWoodHouse();	Yes. Since Object is the class hierarchy of MyMinecraftWoodHouse.
MyMinecraftHouse myHouse = new Object();	No. Since MyMinecraftWoodHouse is not in the class hierarchy of MyMinecraftWoodHouse.

© Tinker Academy 2014

MyMinecraftWoodHouse myWoodHouse = new MyMinecraftHouse();	No. Since MyMinecraftWoodHouse is not in the class hierarchy of MyMinecraftHouse
MyMinecraftHouse myHouse = new MyMinecraftWoodHouse()	Yes. Since MyMinecraftHouse is in the class hierarchy of MyMinecraftWoodHouse

© Tinker Academy 2014

Method Overriding

Method Override
A subclass can define a method that has the same name, signature and return type as that of some method of a class in the class hierarchy.

Such a method definition is said to override the method definition of the super class.

Meh?

Not really!

As you will see below, this allows one of Java's most powerful concepts, **dynamic dispatch**

We will not go into much depth into dynamic dispatch, but suffice to understand that without dynamic dispatch, we would not be able to write MyMinecraftHouse Java Plugin as easily we will be doing today.

The code below displays dynamic dispatch in action.

1. Open Eclipse
2. Select project "StarterPack6" in Package Explorer
3. Open the class StarterPack6.
4. Add the following code in the main method under //TODO (1).

   (Refer to Handout3 for more details on arrays and array access)

   MyMysteryWoodHouseGenerator.createNewHouse creates a new object.
   We don't know the datatype of object (yet).
   All we know is that
   - the datatype of object is some class.
   - the class has MyMinecraftWoodHouse in the class's class hierarchy.
   - the class has the method build()

```
MyMinecraftWoodHouse[] houses = new MyMinecraftWoodHouse[3];
int i = 0;
for (i = 0; i < 3; ++i)
{
    MyMinecraftWoodHouse house = MyMysteryWoodHouseGenerator.createNewHouse();
```

© Tinker Academy 2014

```
    houses[i] = house;
}
```

5. Open Problems View. You should see no Errors.
6. Add the following code in the main method under //TODO (2)
   The code gets the house from the ArrayList and invokes the build method (to build the house).

```
for (i = 0; i < 3; ++i)
{
    MyMinecraftWoodHouse house = houses[i];
    house.build();
}
```

7. Open Problems View. You should see no Errors.
8. The following should be printed to the console

```
Starter Pack 6
building a house (MyMinecraftRedWoodHouse)
building a house house (MyMinecraftBlueWoodHouse)
building a house house (MyMinecraftGreenWoodHouse)
```

9. Open the MyMysteryHouseGenerator source file.

A single source file can have more than 1 class.

MyMysteryHouseGenerator has the various classes

- MyMinecraftRedWoodHouse
- MyMinecraftBlueWoodHouse
- MyMinecraftGreenWoodHouse

All these classes have a different definitions for the build method.

Even though the house variable has a datatype MyMysteryWoodHouse, the Java Runtime could use dynamic dispatch to figure out the correct build() method definition to use.

dynamic dispatch is awesome, because it allows

- Bukkit to create a Java class called JavaPlugin,
- Users to create subclasses of JavaPlugin and override methods defined in the JavaPlugin with their own method definitions

© Tinker Academy 2014

- Bukkit to invoke methods on the subclasses as if it were a JavaPlugin class without being even aware that the subclasses are user defined classes.

The Java Runtime will do the right thing and invoke the correct method definition on the subclass.

How awesome is that!

Identifying the actual class

dynamic dispatch is very cool. However you will need to identify the class for example, that implements the JavaPlugin.

Java provides the **instanceof** operator for this.

The instanceof operator compares the class of an object to another class and returns true if the object's class matches the other class or the object's class is a subclass of the other class.

object	Object's class	is instanceof	Will return
house	MyMinecraftRedWoodHouse	MyMinecraftWoodHouse	true
house	MyMinecraftRedWoodHouse	Object	true
house	MyMinecraftRedWoodHouse	MyMinecraftSandstoneHouse	false

Packages

What is a **package**?

In general, a package is a group of related things. In Java, a package is a group of related classes.

© Tinker Academy 2014

Packages allow better code organization.

Every package have a package name. The name is case sensitive. By convention, packages are in lower case. Package names can have so spaces.

Java allows classes that reside in the same package to access each other fields and methods using the package access modifier.

Every class must reside in a package. If no package is specified, the class is defined to reside in the “default” or “unnamed” package. A class can reside only in one package.

The package is declared using the package keyword and should appear as the first line of code in the Java Source File before the class definition

Package declaration

```
package com.tinkeracademy.java;

public abstract class MyMinecraftHouse {
}
```

Once a class is defined, it is said to “reside in the package” and cannot move to a new package.

If you want to move a class to a new package, you will have to update the class definition. Eclipse provides some powerful tools (called refactor tools) to help with this.

The source file in which the class is defined, must reside in a directory with the same name (case sensitive on Ubuntu Linux). So if a class “resides in a package” called “mypackage”, the source file in which it is defined **must** reside in a directory called “mypackage”.

Java went one step further and has support to create a hierarchy of packages. The parts of the hierarchy are separated by a period (.). This proves very useful for code organization as you build larger libraries.

For example, here are some of the packages in the Java Development Kit (JDK). They are organized in a hierarchy hierarchy. The JDK has 200+ packages.

Packages in Java

Package Name	Package Description

© Tinker Academy 2014

java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Complete Class Names (Refer to Handout3 for details on the classes)

Class	Package	Complete Class Name (including package)
Object	java.lang	java.lang.Object
String	java.lang	java.lang.String
ArrayList	java.util	java.util.ArrayList
HashMap	java.util	java.util.HashMap
HashSet	java.util	java.util.HashSet

Here are the packages in Bukkit. Note that the packages are arranged in a hierarchy. All packages start with “org.bukkit”.

Packages in Bukkit 1.7.10

Package Name	Package Description
org.bukkit	More generalized classes in the API.
org.bukkit.block	Classes used to manipulate the voxels in a world, including special states.
org.bukkit.command	Classes relating to handling specialized non-chat player input.
org.bukkit.command.defaults	Commands for emulating the Minecraft commands and other necessary ones for use by a Bukkit implementation.

© Tinker Academy 2014

org.bukkit.configuration	Classes dedicated to handling a plugin's runtime configuration.
org.bukkit.configuration.file	Classes dedicated facilitating configurations to be read and stored on the filesystem.
org.bukkit.configuration.serialization	Classes dedicated to being able to perform serialization specialized for the Bukkit configuration implementation.
org.bukkit.conversations	Classes dedicated to facilitate direct player-to-plugin communication.
org.bukkit.enchantments	Classes relating to the specialized enhancements to item stacks, as part of the meta data.
org.bukkit.entity	Interfaces for non-voxel objects that can exist in a world, including all players, monsters, projectiles, etc.
org.bukkit.entity.minecart	Interfaces for various Minecart types.
org.bukkit.event	Classes dedicated to handling triggered code executions.
org.bukkit.event.block	Events relating to when a block is changed or interacts with the world.
org.bukkit.event.enchantment	Events triggered from an enchantment table.
org.bukkit.event.entity	Events relating to entities, excluding some directly referencing some more specific entity types.
org.bukkit.event.hanging	Events relating to entities that hang.
org.bukkit.event.inventory	Events relating to inventory manipulation.
org.bukkit.event.painting	Events relating to paintings, but deprecated for more general hanging events.
org.bukkit.event.player	Events relating to players.
org.bukkit.event.server	Events relating to programmatic state changes on the server.
org.bukkit.event.vehicle	Events relating to vehicular entities.
org.bukkit.event.weather	Events relating to weather.

© Tinker Academy 2014

org.bukkit.event.world	Events triggered by various world states or changes.
org.bukkit.generator	Classes to facilitate world generation implementation.
org.bukkit.help	Classes used to manipulate the default command and topic assistance system.
org.bukkit.inventory	Classes involved in manipulating player inventories and item interactions.
org.bukkit.inventory.meta	The interfaces used when manipulating extra data can can be stored inside item stacks.
org.bukkit.map	Classes to facilitate plugin handling of map displays.
org.bukkit.material	Classes that represents various voxel types and states.
org.bukkit.metadata	Classes dedicated to providing a layer of plugin specified data on various Minecraft concepts.
org.bukkit.permissions	Classes dedicated to providing binary state properties to players.
org.bukkit.plugin	Classes specifically relating to loading software modules at runtime.
org.bukkit.plugin.java	Classes for handling plugins written in java.
org.bukkit.plugin.messaging	Classes dedicated to specialized plugin to client protocols.
org.bukkit.potion	Classes to represent various potion properties and manipulation.
org.bukkit.projectiles	Classes to represent the source of a projectile
org.bukkit.scheduler	Classes dedicated to letting plugins run code at specific time intervals, including thread safety.
org.bukkit.scoreboard	Interfaces used to manage the client side score display system.
org.bukkit.util	Multi and single purpose classes to facilitate various programmatic concepts.

© Tinker Academy 2014

org.bukkit.util.io	Classes used to facilitate stream processing for specific Bukkit concepts.
org.bukkit.util.noise	Classes dedicated to facilitating deterministic noise.
org.bukkit.util.permissions	Static methods for miscellaneous permission functionality.

In general, package hierarchy follow reverse domain naming.

The steps below uses Eclipse code refactor tool to move the classes into classes.

We will use the hierarchical package “com.tinkeracademy.java” to organize our classes. All our classes will now move into the “com.tinkeracademy.java” package.

1. Open Eclipse
2. Select project “StarterPack6” in Package Explorer
3. Right Click, Select New Package
4. Type in the package name exactly as shown below. Use the same case. Use the period (.) to separate the words com and tinkeracademy

```
com.tinkeracademy
```

5. For each java source file
    a. Select java source file
    b. Drag the file in Package Explorer into the new package
    c. Alternatively, Right click, Refactor, Move..., select the new package created above, Click OK
6. Check the Problems View. You should not see any errors.
7. Select the source file StarterPack6 (It should be under the new package com.tinker academy). Right click, Run as Java Application.
8. The following should be printed to the console.

```
Starter Pack 6
building a house (com.tinkeracademy.MyMinecraftRedWoodHouse)
building a house house (com.tinkeracademy.MyMinecraftBlueWoodHouse)
building a house house (com.tinkeracademy.MyMinecraftGreenWoodHouse)
```

© Tinker Academy 2014

Classpath

What is a classpath?

Classpath is the mechanism Java uses to locate Java source files and Java compiled files (byte code).

Its important to understand that the Java Compiler and the Java Runtime need to be able to locate packages. Remember that a package name maps to a directory name or a sub path or the file system. But the Java Compiler and the Java Runtime need to know the base location(s) from which to start searching for Java source files (Java Compiler) or the Java class files (Java Compiler and Java Runtime). The classpath defines the list of possible locations to search for a Java Source File or a Java Class File.

Interfaces

What is an Interface?

A typical object oriented project defines hundreds of classes.

Each class will declare a number of methods. The method declarations form the behavior of the class. Different classes may implement the same behavior in different ways.

The behavior can be grouped into a logical set of interfaces. An interface is just a logical set of behaviors that a class implements.

An interface just specifies the methods. A class can then implement the interface.
In the code below, we will define a new interface call IHouse.

The interface defines the behavior of a house.

The first capital letter I is a convention we will follow to indicate this source file contains an interface.

As of now, the interface has just one method called build.

1. Open Eclipse
2. Select project “StarterPack6” in Package Explorer

© Tinker Academy 2014

3. Right Click, select New Interface. Type in

Package	com.tinkeracademy
Name	IHouse

4. Click Finish to create the interface. A new source file IHouse.java will be created under src/com/tinkeracademy
5. Add the following method declaration in the interface IHouse

```
public void build();
```

Note that just the method declaration is specified in the interface.

6. Open the source file MyMinecraftHouse.java
7. Modify the class signature to

```
public abstract class MyMinecraftHouse implements IHouse
```

The class implements the behavior specified in com.tinkeracademy.IHouse

7. Open Problems View. You should not see any errors.
8. Save your changes

An Interfaces can extend another interface similar to that of a Java Class. The interface being extended from is called the superinterface.

For example, in Bukkit, the interface org.bukkit.plugin.Plugin interface extends the interface org.bukkit.command.TabExecutor. Here org.bukkit.command.TabExecutor is the superinterface.

A class can implement multiple interfaces.

Complete the Plugin!

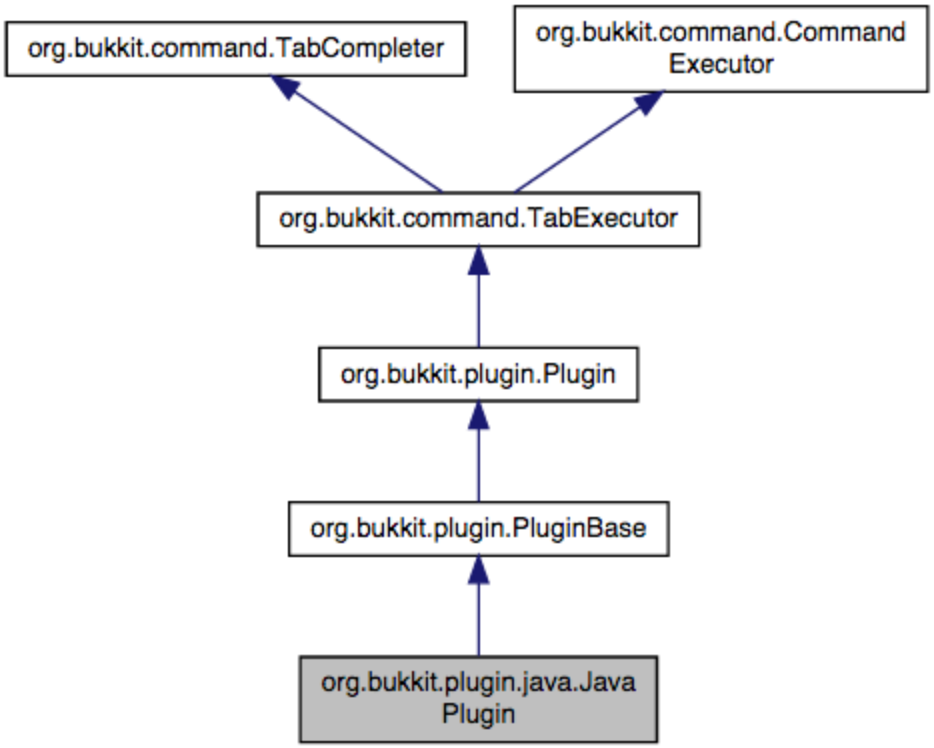
The following steps will complete the Minecraft Plugin. The plugin will respond to various commands to create different kinds of Minecraft Houses.

1. Open Eclipse
2. Select project “StarterPack6” in Package Explorer

© Tinker Academy 2014

3. Open source file StarterPack6.java
4. The class StarterPack6 will extend JavaPlugin.
   JavaPlugin is a Bukkit class in the package org.bukkit.plugin.java
   JavaPlugin represents a (surprise!) Java Plugin

   The UML Diagram shows the class hierarchy for com.bukkit.plugin.java.JavaPlugin



   Class PluginBase is a superclass of JavaPlugin.
   PluginBase implements the interface Plugin.
   TabExecutor is the superinterface of the interface Plugin.
   TabCompleter is one of the superinterfaces of TabExecutor.
   CommandExecutor is the other superinterface of TabExecutor.

   Change the class signature of StarterPack6 to the following

```
public class StarterPack6 extends JavaPlugin
```

5. Fix any import issues that show up as red markers in Eclipse.
6. StarterPack6 will override the onCommand method defined in JavaPlugin.

   Right Click on StarterPack6. Select Source. Select Override/Implement Methods…

© Tinker Academy 2014

Click Deselect All. Select onCommand from the list.

```
public boolean onCommand(CommandSender sender, Command command, String
label, String[] args) {
	//TODO Auto-generated method stub
	return super.onCommand(sender, command, label, args);
}
```

The method onCommand accepts the following inputs

Input Name	DataType	Why is this required?
sender	CommandSender	Used to indicate who sent the command
command	Command	Represents the Command that the user sent
label	String	command label that user types in
args	String[] (string array) Refer to Handout3 for details on String[]	any additional arguments

7. Fix any import issues
8. Open Problems View. You should not see any errors.
9. Remove the auto generated code from the method definition and add the code below

```
return false;
```

10. After removing the code the method should look like below.

```
public boolean onCommand(CommandSender sender, Command command, String
label, String[] args) {
	return false;
}
```

11. Add the following method definition code to onCommand

The code does the following

© Tinker Academy 2014

1. verifies that the label (which is of datatype String) is same as “buildwoodhouse” by invoking the equalsIgnoreCase method on the label. equalsIgnoreCase is a method declared by the String class. The method compares the object with another string and returns true if the strings are the same after ignoring the case (label.equalsIgnoreCase(“BUILDWOODHOUSE”) will return true.
2. uses instanceof to verify that the Player sent the command
3. creates a new MyMinecraftWoodHouse and assigns it to the variable house
4. invokes the build method on house
5. returns true if the the steps above are successful

```
if (label.equalsIgnoreCase("buildwoodhouse")) {
      if (sender instanceof Player) {
        Player player = (Player) sender;
        Location location = player.getLocation();
        MyMinecraftHouse house = new MyMinecraftWoodHouse(“Casa Rosa”, 10, 10,
location);
        house.build();
        return true;
        }
}
return false;
```

10. Fix any import issues
11. Open Problems View. You should not see any errors.

We will build the house and package the plugin into Minecraft in the next class!

That was a LOT we covered!

You made it this far! Awesome!

© Tinker Academy 2014

We will cover **Build a House Plugin (Minecraft Plugin Development)** in the next class class. For now you need to make sure you have a good conceptual understand of Class Inheritance, Packages and Interfaces in **Inheritance, Packages and Interfaces Part 2.**

© Tinker Academy 2014

Quiz 6: Inheritance, Packages and Interfaces Part 2

Make sure you read this Handout!

Open the Quiz

Make sure you are on the Home WiFi.
Follow the instructions in “Updating the Course” in this Handout.
Open Quiz6.odt under “Courses” “TA-JAV-1” “quiz” “quiz6”

Complete the Quiz

1. Attempt each question. Type in the answers in the “Answer:” box.
2. Save the file using File->Save or Ctrl-S

Submit the Quiz

Make sure you are on the Home WiFi.
Follow the instructions in “Submitting Homework” in this Handout.

© Tinker Academy 2014

Homework 6: Inheritance, Packages and Interfaces Part 2

Make sure you read this Handout!

Overview

In this Homework you will add a new helper method to make a cuble of blocks of a specified type in Minecraft.

Open the Homework

Follow the instructions in “Updating the Course” in this Handout.
Extract Homework6.zip under “Courses” “TA-JAV-1” “homework” “homework6”

- Select Homework zip file
- Right Click, Select Extract Here

Import the Homework6 project in Eclipse

- Right Click over Project Navigator
- Select Import Project
- Toggle on General, Existing Projects into Workspace
- Browse to the Homework6 folder
- Click OK

Complete the Homework

You will need to refer to this handout (Handout5). Make sure you read it thoroughly.

Before You Begin

Structure of the Program
Homework6 is an Eclipse Java Project. The Project has

1. A Java Source File called “Homework6.java”
2. A Java Source File called “MyMinecraftHouse.java”
3. A Java Source File called “MyMinecraftWoodHouse.java”
4. Project references to the Java Runtime Environment 1.6 System Library.
5. Project references to the Minecraft Plugin Library.

© Tinker Academy 2014

Run the Program

1. Click over project name “Homework6”.
2. Toggle the src folder to select Homework6.java
3. Right Click. Select “Run As…”. Select “Java Application”.
   The text “Homework 6” should be displayed in the Console window (bottom)

Homework Steps

1. Add a new method to the class MyMinecraftHouse. The method should have the following signature

```
private void makeCube(Location loc, int width, int height, Material mat) {
    //TODO complete the implementation
}
```

2. Complete the implementation of makeCube

```
Location blockLoc = new Location(loc.getWorld(), 0, 0, 0);
int i;
int j;
int k;

for (i = 0; i < width; i++) {
   for (j = 0; j < width; j++) {
      for (k = 0; k < height; k++) {
         blockLoc.setX(loc.getX() + i);
         blockLoc.setY(loc.getY() + j);
         blockLoc.setZ(loc.getZ() + k);
         Block block = loc.getWorld().getBlockAt(blockLoc);
         block.setType(mat);
      }
   }
}
```

3. Save your changes.
4. Open Problems View. You should not see any errors.

Submit the Homework

© Tinker Academy 2014

Make sure you are on the Home WiFi.
Follow the instructions in “Submitting Homework” in this Handout.

© Tinker Academy 2014