



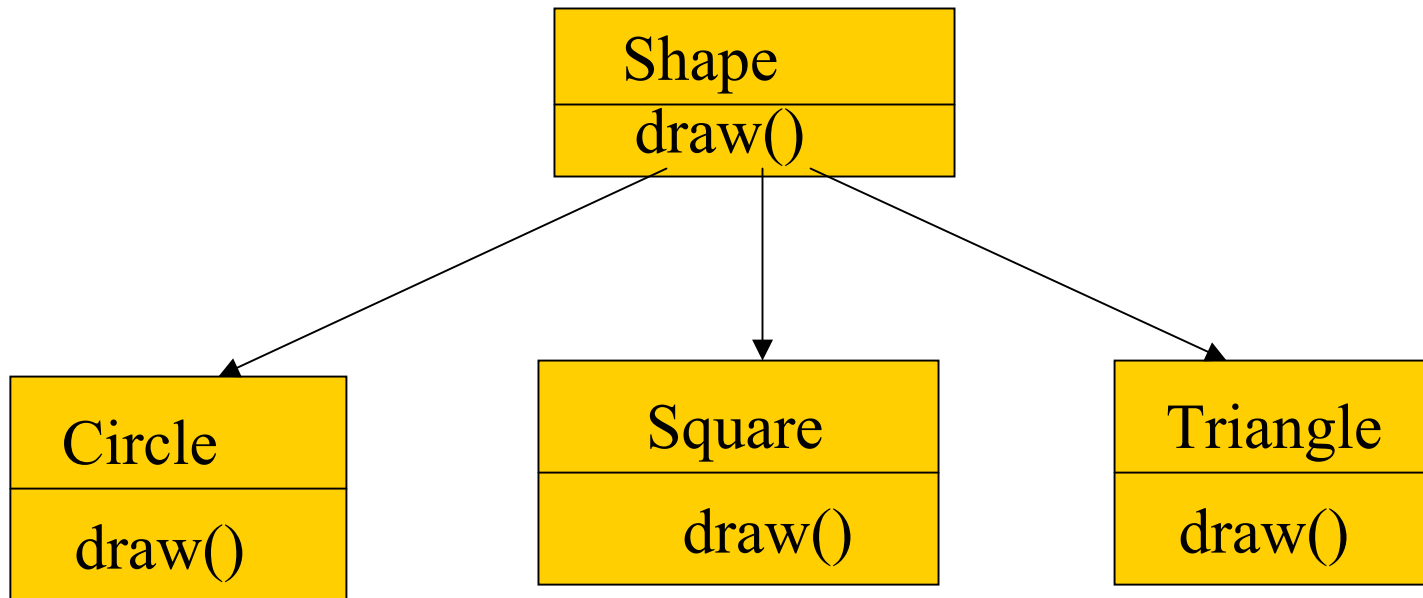
Run-time Type Identification

- This module looks at the ways that Java allows you to discover information about objects and classes at run-time. This takes two forms: “traditional” RTTI, which assumes that you have all the types available at compile-time and run-time, and the “reflection” mechanism, which allows you to discover class information solely at run-time. The “traditional” RTTI will be covered first, followed by a discussion of reflection.



The need for RTTI

- Consider the now familiar example of a class hierarchy that uses polymorphism.





The need for RTTI

- `import java.util.*;`
- `class Shape1 {`
- `void draw() {`
- `System.out.println("I am in Shape draw, I can draw nothing"); }`
- `}`
- `class Circle1 extends Shape1 {`
- `public void draw() {`
- `System.out.println("I am in Circle draw(), I can draw Circle here");`
- `}`
- `}`
- `class Square1 extends Shape1 {`
- `public void draw() {`
- `System.out.println("I am in Square draw(), I can draw Square here");`
- `}`
- `}`



The need for RTTI

- `class Triangle1 extends Shape1 {`
- `public void draw() {`
- `System.out.println("I am in Triangle draw(), I can draw Triangle here");`
- `}`
- `}`
- `public class Shapes1 {`
- `public static void main(String[] args) {`
- `ArrayList s = new ArrayList();`
- `s.add(new Circle1());`
- `s.add(new Square1());`
- `s.add(new Triangle1());`
- `Iterator e = s.iterator();`
- `while(e.hasNext())`
- `((Shape1)e.next()).draw();`
- `}`
- `} ///:~`



The need for RTTI

- The following is the next version of Shape, Try to understand the code
- `import java.util.*;`
- `class Shape {`
- `void draw() {`
- `System.out.println(this + ".draw()");`
- `}`
- `}`
- `class Circle extends Shape {`
- `public String toString() {`
- `return "Circle"; }`
- `}`
- `class Square extends Shape {`
- `public String toString() {`
- `return "Square"; }`
- `}`



The need for RTTI

- `class Triangle extends Shape {`
- `public String toString() {`
- `return "Triangle";`
- `}`
- `}`
- `public class Shapes {`
- `public static void main(String[] args) {`
- `ArrayList s = new ArrayList();`
- `s.add(new Circle());`
- `s.add(new Square());`
- `s.add(new Triangle());`
- `Iterator e = s.iterator();`
- `while(e.hasNext()) ((Shape)e.next()).draw();`
- `}`
- `} ///:~`

Written by Paul Pu All Rights
Reserved www.torontocollege.com



The Class object

- To understand how RTTI works in Java, you must first know how type information is represented at run-time. This is accomplished through a special kind of object called the *Class object*, which contains information about the class. (This is sometimes called a *meta-class*.) In fact, the **Class** object is used to create all of the “regular” objects of your class.



The Class object

- There's a **Class** object for each class that is part of your program. That is, each time you write and compile a new class, a single **Class** object is also created (and stored, appropriately enough, in an identically named **.class** file). At run-time, when you want to make an object of that class, the Java Virtual Machine (JVM) that's executing your program first checks to see if the **Class** object for that type is loaded. If not, the JVM loads it by finding the **.class** file with that name. Thus, a Java program isn't completely loaded before it begins, which is different from many traditional languages.
- Once the **Class** object for that type is in memory, it is used to create all objects of that type.



The Class object

- If this seems shadowy or if you don't really believe it, here's a demonstration program to prove it:
// Examination of the way the class loader works.
- `class Candy {`
- `static { System.out.println("Loading Candy");`
- `}`
- `}`
- `class Gum {`
- `static { System.out.println("Loading Gum");`
- `}`
- `}`
- `class Cookie {`
- `static { System.out.println("Loading Cookie");`
- `}`
- `}`



The Class object

- `public class SweetShop {`
- `public static void main(String[] args) {`
- `System.out.println("inside main");`
- `new Candy();`
- `System.out.println("After creating Candy");`
- `try { Class.forName("Gum"); }`
- `catch(ClassNotFoundException e) { e.printStackTrace(); }`
`System.out.println("After Class.forName(\"Gum\")");`
- `new Cookie();`
- `System.out.println("After creating Cookie");`
- `}`
- `} ///:~`



The Class object

- A particularly interesting line is:
- `Class.forName("Gum");`
- This method is a **static** member of **Class** (to which all **Class** objects belong). A **Class** object is like any other object and so you can get and manipulate a reference to it. (That's what the loader does.) One of the ways to get a reference to the **Class** object is **forName()**, which takes a **String** containing the textual name (watch the spelling and capitalization!) of the particular class you want a reference for. It returns a **Class** reference.



The Class object

- The output of this program for one JVM is:
 - inside main
 - Loading Candy
 - After creating Candy
 - Loading Gum
 - After `Class.forName("Gum")`
 - Loading Cookie
 - After creating Cookie
 - You can see that each **Class** object is loaded only when it's needed, and the **static** initialization is performed upon class loading.



RTTI syntax

- **RTTI syntax**
- **Java performs its RTTI using the Class object, even if you're doing something like a cast. The class Class also has a number of other ways you can use RTTI.**
- **First, you must get a reference to the appropriate Class object. One way to do this, as shown in the previous example, is to use a string and the Class.forName() method. This is convenient because you don't need an object of that type in order to get the Class reference. However, if you do already have an object of the type you're interested in, you can fetch the Class reference by calling a method that's part of the Object root class: getClass(). This returns the Class reference representing the actual type of the object. Class has many interesting methods, demonstrated in the following example: ToyTest.java**



Reflection: run-time class information

java.lang.reflect package

- Field class**
- Method class**
- Constructor class**
- Modifier class**