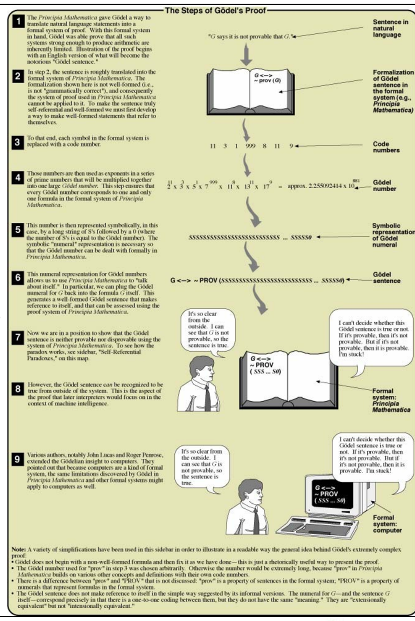
Inheritance and Godel's Proof

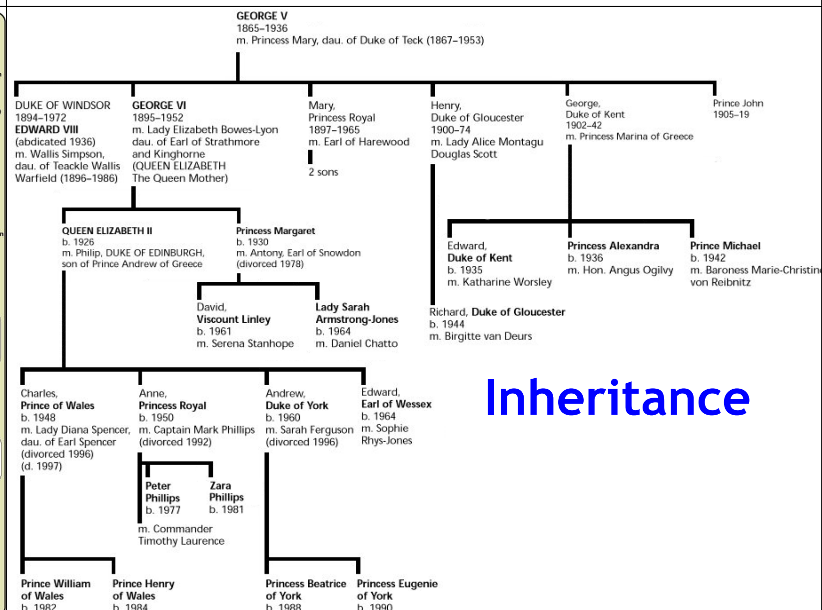
One-Slide Summary

- **Inheritance** allows a subclass to share behavior (methods and instance variables) with a superclass.
- A **class hierarchy** shows how subclasses inherit from superclasses. Typically a single ultimate class, such as ***object***, lies at the top of a class hierarchy.
- A subclass can be used whenever a superclass is expected. This is called **subtyping**.
- An **axiomatic system** provides a way to reason **mechanically** about formal notions. An **incomplete** system fails to prove some true statements. An **inconsistent** system proves some false statements.
- ***Any*** interesting logical system is incomplete: there is a true statement that cannot be proved in it.

Outline

- Inheritance
- Subtyping
- PS6
- Mechanical Reasoning
- Axiomatic Systems
- Paradoxes
- Gödel





Inheritance

Hey, Scooby!

```
public class Dog {
  public Dog(String n) { name = n; }
  public String name;
  public void bark() { println("wuff wuff"); } }
public class TalkingDog extends Dog {
  public void speak(String words) {
    println(name + " says " + words);
  } } // inherits all Dog fields and methods

Dog judy = new Dog("Judy");
judy.bark();
wuff wuff
judy.speak("salve atque vale!");
Type Error!
TalkingDog scooby = new TalkingDog("Scooby");
scooby.bark();
wuff wuff
scooby.speak("solve the mystery!");
Scooby says solve the mystery!
```


Overriding Inherited Behavior

```
public class Dog {
  public Dog(String n) { name = n; }
  public String name;
  public void bark() { println("wuff wuff"); } }
public class TalkingDog extends Dog {
  public void bark() { println("wuff wuff, but I could speak!"); }
  public void speak(String words) {
    println(name + " says " + words);
  } }

Dog judy = new Dog("Judy");
judy.bark();
wuff wuff
TalkingDog scooby = new TalkingDog("Scooby");
scooby.bark();
wuff wuff, but I could speak!
scooby.speak("solve the mystery!");
Scooby says solve the mystery!
```

Dynamic (= Run-Time) Types

```
public class Dog {
  public Dog(String n) { name = n; }
  public String name;
  public void bark() { println("wuff wuff"); } }
public class TalkingDog extends Dog {
  public void bark() { println("wuff wuff, but I could speak!"); }
  public void speak(String words) {
    println(name + " says " + words);
  } }

Dog judy = new Dog("Judy");
TalkingDog scooby = new TalkingDog("Scooby");
Dog myDog;          // static type myDog = Dog
myDog = judy;       // dynamic type MyDog = Dog
myDog.bark();
wuff wuff
myDog = scooby;  // dynamic type myDog = TalkingDog
myDog.bark();
wuff wuff, but I could speak!
```

#7

Static (= Compile-Time) Types

```
public class Dog {
  public Dog(String n) { name = n; }
  public String name;
  public void bark() { println("wuff wuff"); } }
public class TalkingDog extends Dog {
  public void bark() { println("wuff wuff, but I could speak!"); }
  public void speak(String words) {
    println(name + " says " + words);
  } }

Dog judy = new Dog("Judy");
TalkingDog scooby = new TalkingDog("Scooby");
Dog myDog;          // static type myDog = Dog
myDog = judy;       // dynamic type MyDog = Dog
myDog.speak("i am the very model");
Type Error: Dog has no method speak()
myDog = scooby;  // dynamic type myDog = TalkingDog
myDog.speak("of a modern major-general");
Type Error: Dog has no method speak()
```

#8

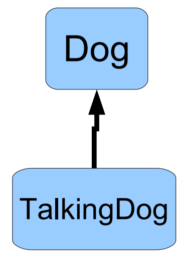
Types and Objects

- Whether or not you can even ***call*** a method depends on the **static type** of the object!
  - The static type is the declared type of the variable in the source code:
    - Dog myDog = new TalkingDog(); // static type Dog
    - myDog.speak("hello"); // Type Error
- The exact ***behavior*** of a method depends on the **dynamic type** of the object!
  - The dynamic type is **X** if you wrote **new X()**:
    - Dog myDog = new TalkingDog(); // dynamic TalkingDog
    - myDog.bark(); // wuff wuff, but I could speak!

#9

Speaking About Inheritance

- Inheritance is using the definition of one class to define another class.
- TalkingDog **inherits** from Dog.
- TalkingDog is a **subclass** of Dog.
- The **superclass** of TalkingDog is Dog.
- *These all mean the same thing!*



#10

Subtyping

- **Subtyping:** An object of a subclass can be used whenever an object of a superclass is expected.
  - Intuition: The subclass will only ever have "more stuff", so this is safe. Inheritance can only add new behavior or change old behavior.
  - This is called the Barbara Liskov Substitution Principle.
- "Dog myDog = new TalkingDog();" works because TalkingDog is a subclass of Dog!

#11

Subtyping Exercise

```
public class Animal { ...getMass()... }
public class Dog extends Animal { ..bark().. }
public class TalkingDog extends Dog { ..speak().. }
void myMethod(Animal a, Dog d, TalkingDog t) { // Which are valid?
  Animal zooPet = a; zooPet.getMass();
  zooPet = d; zooPet.getMass();
  zooPet = t; zooPet.getMass();
  Dog myDog = a; myDog.bark();
  myDog = d; myDog.bark();
  myDog = t; myDog.bark();
  TalkingDog cartoon = a; cartoon.speak();
  cartoon = d; cartoon.speak();
  cartoon = t; cartoon.speak();
```

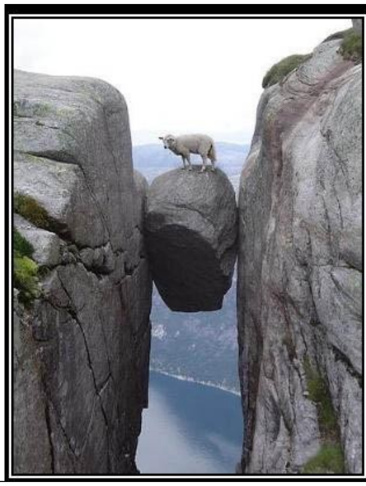
#12

Subtyping Exercise

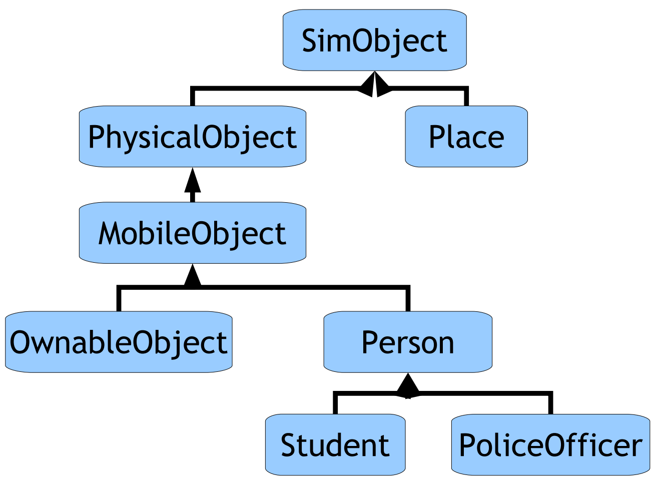
public class Animal { ...getMass()... }
public class Dog extends Animal { ..bark().. }
public class TalkingDog extends Dog { ..speak().. }
void myMethod(Animal a, Dog d, TalkingDog t) { // Which are valid?
  Animal zooPet = a; zooPet.getMass();
  zooPet = d; zooPet.getMass();
  zooPet = t; zooPet.getMass();
  ~~Dog myDog = a; myDog.bark();~~
  myDog = d; myDog.bark();
  myDog = t; myDog.bark();
  ~~TalkingDog cartoon = a; cartoon.speak();~~
  ~~cartoon = d; cartoon.speak();~~
  cartoon = t; cartoon.speak();

Problem Set 6

- Make an adventure game by programming with objects.
- Many objects in our game have similar properties and behaviors, so we use inheritance.



PS6 Classes



Object-Oriented Terminology

- An **object** is an entity that packages state and procedures.
- A **constructor** is a procedure that creates new objects.
- The state variables that are part of an object are called **instance variables**.
- The procedures that are part of an object are called **methods**. We **invoke** (call) a method.
- **Inheritance** allows one class to refine and reuse the behavior of another.
- **Subtyping** allows a subclass to be used where a superclass is expected.

Review-ish: Python Dictionaries

- The **dictionary** abstraction provides a lookup table.
- Each entry is a pair:
  <*key*, *value*>
- The *key* must be an immutable object. The *value* can be anything.
- *dictionary*[*key*] evaluates to the *value* associated with *key*
  - Running time is approximately constant!
  - (e.g., “associative array” or “hash table”)

Dictionary Example

```
>>> d = {}                 # new empty dictionary
>>> d['UVA'] = 1818        # make new entry
>>> d['UVA'] = 1819        # update entry
>>> d['Cambridge'] = 1209
>>> d['UVA']
1819
>>> d['Oxford']
KeyError: 'Oxford'
```

Pencil & Paper: Histograms

- Define a procedure **histogram** that takes a text string as input. The procedure returns a dictionary in which each word in the input string is mapped to the number of times it occurs in that string.
- Hints:
  - Iterate over each word, putting it in a dictionary. If you haven't seen it before, its count is 1. Otherwise, increment its count.

>>> 'here we go'.split()

['here', 'we', 'go']

Histogram Example

```
def histogram(text):
  d = {}
  words = text.split()
  for w in words:
    if w in d:
      d[w] = d[w] + 1
    else:
      d[w] = 1
  return d
```

```
>>> d = histogram(declaration)
>>> show_dictionary(d)
of: 79
the: 76
to: 64
and: 55
...
```


Java HashMaps

- We can do the same thing in Java:

```
d = {}              # new empty dictionary
d['UVA'] = 1818     # make new entry
d['UVA']            # get a value
1819
```

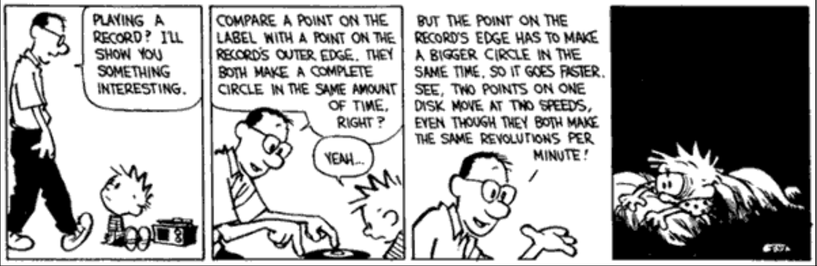
```
HashMap<String, Integer> d = new HashMap<String, Integer>;
d.put("UVA", 1818);
d.get("UVA");
1819
```


Author Fingerprinting

- [...] a comparison of phrases used in The Reign of King Edward III with Shakespeare's early works proves conclusively that the Bard wrote the play in collaboration with Thomas Kyd, one of the most popular playwrights of his day. [...] He discovered that playwrights often use the same patterns of speech, meaning that they have a linguistic fingerprint. The program identifies phrases of three words or more in an author's known work and searches for them in unattributed plays. In tests where authors are known to be different, there are up to 20 matches because some phrases are in common usage. When Edward III was tested against Shakespeare's works published before 1596 there were 200 matches.
  - Jack Malvern, "Computer program proves Shakespeare didn't work alone, researchers claim", *The Times of London*, 12 Oct 2009

Liberal Arts Trivia: Physics

- Name the vector quantity in physics measured in radians per second. The direction of the vector is perpendicular to the plane of rotation and is usually specified by the "right hand rule".



Liberal Arts Trivia: Chemistry

- Give the common name for hydragyrum, a heavy metal element. It is the only element that is liquid at standard temperature and pressure and is often used in the construction of sphygmomanometers. In the 18^{th} to 19^{th} centuries it was used to make felt hats, and the psychological symptoms associated with its poisoning are sometimes used to explain the phrase "mad as a hatter".
- Bonus: What does a sphygmomanometer measure?

Upcoming Deadlines

- Start PS6 early
  - PS6 is challenging
  - Opportunity for creativity
- Start thinking about PS9 Project ideas
  - If you want to do an "extra ambitious" project convince me your idea is worthy before (ps7 and 8) or (ps8)
  - Discuss ideas and look for partners **on the forum**

#25

Story So Far

- Much of the course so far:
  - Getting comfortable with recursive definitions
  - Learning to write a program to do (almost) anything (PS1-4)
  - Learning more elegant ways of programming (PS5-6)
- This Week:
  - Getting ***un***-comfortable with recursive definitions
  - Understanding why there are some things no program can do!

#26

Computer Science/Mathematics

- Computer Science (Imperative Knowledge)
  - Are there (well-defined) problems that cannot be solved by *any* procedure?

Today

- Mathematics (Declarative Knowledge)
  - Are there true conjectures that cannot be the shown using *any* proof?

#27

Mechanical Reasoning

Aristotle (~350BC): *Organon*

Codify logical deduction with rules of inference (syllogisms)

Every A is a P
X is an A — Premises
X is a P — Conclusion

Every *human* is *mortal.*
Gödel is *human.*
Gödel is *mortal.*

#28

More Mechanical Reasoning

- Euclid (~300BC): *Elements*
  - We can reduce geometry to a few axioms and derive the rest by following rules
- Newton (1687): *Philosophiæ Naturalis Principia Mathematica*
  - We can reduce the motion of objects (including planets) to following axioms (laws) mechanically

#29

Mechanical Reasoning

- Late 1800s – many mathematicians working on codifying "laws of reasoning"
  - George Boole, *Laws of Thought*
  - Augustus De Morgan
- Whitehead and Russell, 1911-1913
  - *Principia Mathematica*
  - Attempted to formalize all mathematical knowledge about numbers and sets

#30

All **true** statements about numbers

Perfect Axiomatic System

Derives **all** true statements, and **no** false statements starting from a finite number of axioms and following mechanical inference rules.

Incomplete Axiomatic System

incomplete

Derives **some, but not all** true statements, and **no** false statements starting from a finite number of axioms and following mechanical inference rules.

Inconsistent Axiomatic System

Derives **all** true statements, and **some** false statements starting from a finite number of axioms and following mechanical inference rules.

also derives some false statements

Principia Mathematica

- Whitehead and Russell (1910– 1913)
  - Three Volumes, 2000 pages
- Attempted to axiomatize mathematical reasoning
  - Define mathematical entities (like numbers) using logic
  - Derive mathematical “truths” by following mechanical rules of inference
  - Claimed to be ***complete*** and ***consistent***
    - All true theorems could be derived
    - No falsehoods could be derived

Russell’s Paradox

- Some sets are not members of themselves
  - set of all Students
- Some sets are members of themselves
  - set of all things that are not Students
- *S* = **the set of all sets that are not members of themselves**
- Is *S* a member of itself?

Russell's Paradox

- *S* = set of all sets that are not members of themselves
- Is *S* a member of itself?
  - If *S* **is** an element of *S*, then *S* **is** a member of itself and should **not** be in *S*.
  - If *S* **is not** an element of *S*, then *S* **is not** a member of itself, and **should** be in *S*.

This is Problematic

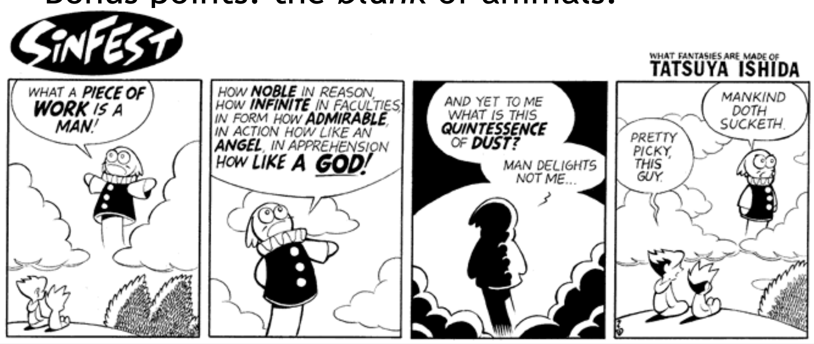
- PM to be ***complete*** and ***consistent***
  - All true theorems could be derived
  - No falsehoods could be derived
- Russel's Paradox is either (true + not derived) or (false + derived).

Ban Self-Reference?

- *Principia Mathematica* attempted to resolve this paragraph by banning self-reference
- Every set has a **type**
  - The lowest type of set can contain only "objects", not "sets"
  - The next type of set can contain objects and sets of objects, but not sets of sets

Liberal Arts Trivia: English Literature and Drama

- Name the tragedy by Shakespeare parodied below by Tatsuya Ishida.
- Bonus points: the *blank* of animals.

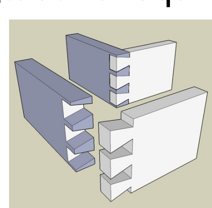


Liberal Arts Trivia: American Law

- This 1925 Tennessee trial was an American legal case that tested the *Butler Act*, which made it unlawful "to teach any theory that denies the story of the Divine Creation of man as taught in the Bible, and to teach instead that man has descended from a lower order of animals" in any Tennessee state-funded school and university. The trial was a wastershed in American creation-evolution controversy. The trial involved two celebrity lawyers, William Jennings Bryan for the prosecution and Clarence Darrow for the defense, and was followed on radio in America.
- Bonus points: What was the outcome?

Liberal Arts Trivia: Woodworking

- This woodworking joinery technique is noted for its tensile strength (resistance to being pulled apart). A series of *pins* are cut from the end of one board and interlock with a series of *tails* cut into the end of another. Once glued it requires no fasteners.



Russell's Resolution?

Set ::= Set_n

Set_0 ::= { x | x is an *Object* }

Set_n ::= { x | x is an *Object* or a $Set_{\underline{n-1}}$ }

S: Set_n

Is S a member of itself?

#43

Russell's Resolution?

Set ::= Set_n

Set_0 ::= { x | x is an *Object* }

Set_n ::= { x | x is an *Object* or a $Set_{\underline{n-1}}$ }

S: Set_n

Is S a member of itself?

No, it is a Set_n so, it can't be a member of a Set_n

#44

Epimenides Paradox

Epimenides (a Cretan):
"All Cretans are liars."

Equivalently:
"This statement is false."

Russell's types can help with the set paradox, but not with these.

#45

Gödel's Solution

All consistent axiomatic formulations of number theory include *undecidable* propositions.

(GEB, p. 17)

undecidable – cannot be proven either true or false inside the system.

#46

Kurt Gödel

- Born 1906 in Brno (now Czech Republic, then Austria-Hungary)
- 1931: publishes *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme* (*On Formally Undecidable Propositions of Principia Mathematica and Related Systems*)

#47

- 1939: flees Vienna
- Institute for Advanced Study, Princeton
- Died in 1978 – convinced everything was poisoned and refused to eat

#48

Gödel's Theorem

In the *Principia Mathematica system*, there are statements that cannot be proven either true or false.

Gödel's Theorem

In any interesting rigid system, there are statements that cannot be proven either true or false.

Gödel's Theorem

All logical systems of any complexity are **incomplete**: there are statements that are *true* that cannot be proven within the system.

Proof – General Idea

- Theorem: In the Principia Mathematica system, there are statements that cannot be proven either true or false.
- Proof: Find such a statement!

Gödel's Statement

G: This statement does not have any proof in the system of *Principia Mathematica*.

G is unprovable, but true!

Why?

Gödel's Proof Idea

G: This statement does not have any proof in the system of *PM*.

If *G* is provable, PM would be inconsistent.

If *G* is unprovable, PM would be incomplete.

Thus, **PM cannot be complete and consistent!**

Homework

- Read Chapter 11
- PS6 Due Soon