Inheritance, Memoization
What is inheritance?

Write a definition of “inheritance”, the computer-science concept. It might be helpful to think about why inheritance is useful: why do programming languages have inheritance as a feature?

(Your response)
Graph code
Java interface

What can graphs do?

```java
import java.util.Set;

public interface DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    // add, remove, and access EDGES
    public EdgeDataType getEdge(NodeDataType srcNode, NodeDataType dstNode);
    public boolean adjacent(NodeDataType srcNode, NodeDataType dstNode);
    public boolean addEdge(NodeDataType srcNode, NodeDataType dstNode, EdgeDataType edge);
    public EdgeDataType removeEdge(NodeDataType srcNode, NodeDataType dstNode);

    // access, add, and remove NODES
    public boolean containsNode(NodeDataType nodeData);
    public Set<NodeDataType> getNodes();
    public Set<NodeDataType> neighbors(NodeDataType srcNode); // adjacent nodes
    public boolean addNode(NodeDataType nodeData);
    public boolean removeNode(NodeDataType nodeData);
}
```
Java implementations (classes)

*How do graphs do it?*

```java
public class EdgeList<NodeDataType, EdgeDataType> implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {
    ...
}
```

```java
public class AdjacencyList<NodeDataType, EdgeDataType> implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {
    ...
}
```

```java
public class AdjacencyMatrix<NodeDataType, EdgeDataType> implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {
    ...
}
```
Edge list in Java

An edge stores source, information, and destination.
A graph stores a list of edges.

```java
public class EdgeList<NodeDataType, EdgeDataType>
    implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    /** private, inner class to represent an edge */
    private class Edge {
        NodeDataType source;
        NodeDataType destination;
        EdgeDataType data;
    }

    /** the edge list */
    private Collection<Edge> theGraph = new ArrayList<Edge>();

    /** keeps track of all nodes (including disconnected ones) */
    private Set<NodeDataType> allNodes = new HashSet<NodeDataType>();

    // ...
public class AdjacencyList<NodeDataType, EdgeDataType>
    implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    /** private, inner class to represent an edge */
    private class Edge {
        NodeDataType source;
        NodeDataType destination;
        EdgeDataType data;
        ...
    }

    /** the edge list */
    private Collection<Edge> theGraph = new ArrayList<Edge>();

    /** keeps track of all nodes (including disconnected ones) */
    private Set<NodeDataType> allNodes = new HashSet<NodeDataType>();

    /** private, inner class to represent an adjacency */
    private class Adjacency {
        NodeDataType destination;
        EdgeDataType data;
        ...
    }

    /** the adjacency list */
    Map<NodeDataType, Collection<Adjacency>> theGraph =
        new HashMap<NodeDataType, Collection<Adjacency>>();

    ...
}

An adjacency stores edge data and destination.
A graph stores a map from nodes to adjacencies.
Adjacency matrix in Java

A graph stores an N x N table.
where N is the number of nodes, rows are sources, columns are destinations

A cell in the table stores the weight of row → column edge.
if there is no edge from row to column, the cell’s weight is ∞

```java
public class AdjacencyMatrix<NodeDataType, EdgeDataType>
    implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    /** the adjacency matrix */
    private EdgeDataType[][] theGraph = new EdgeDataType[0][0];

```
Adjacency matrix in Java

A graph stores an N x N table.
where N is the number of nodes, rows are sources, columns are destinations

A cell in the table stores the weight of row → column edge.
if there is no edge from row to column, the cell’s weight is ∞

```java
public class AdjacencyMatrix<NodeDataType, EdgeDataType>
    implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    /** the adjacency matrix */
    private Object[][] theGraph = new Object[0][0];
```

Adjacency matrix in Java

A graph stores an N x N table. where N is the number of nodes, rows are sources, columns are destinations.

A cell in the table stores the weight of row → column edge. if there is no edge from row to column, the cell’s weight is ∞.

```java
public class AdjacencyMatrix<NodeDataType, EdgeDataType>
    implements DirectedWeightedGraph<NodeDataType, EdgeDataType> {

    /** the adjacency matrix */
    private Object[][] theGraph = new Object[0][0];

    /** allows us to convert a node's data to an array index */
    private Map<NodeDataType, Integer> nodeIndexLookup =
        new HashMap<NodeDataType, Integer>();

    . . .
}
```
Why inheritance?
Modeling the World with Objects & Classes

Buddy, Age 5
Woof

Daisy, Age 3
Woof

Zoe, Age 1
Woof
47 spots

Howard, Age 2
Woof
42 spots

Tiger, Age 5
Meow
The Dog class

Buddy, Age 5

Woof

Daisy, Age 3

Woof

Dog buddy = new Dog("Buddy", 5);
Dog daisy = new Dog("Daisy", 3);

```java
public class Dog {
    String myName;
    int myAge;

    public Dog(String n, int a) {
        this.myName = n;
        this.myAge = a;
    }

    public void sayHello() {
        System.out.println("Woof");
    }
}
```
The Dalmatian class

Zoe, Age 1
47 spots

Howard, Age 2
42 spots

Dalmatian zoe = new Dalmatian("Zoe", 1, 47);
Dalmatian howard = new Dalmatian("Howard", 2, 42);

```java
public class Dalmatian {
    String myName;
    int myAge;
    int mySpots;

    public Dalmatian(String n, int a, int s) {
        this.myName = n;
        this.myAge = a;
        this.mySpots = s;
    }

    public void sayHello() {
        System.out.println("Woof");
    }
}
```
Code smell

From Wikipedia, the free encyclopedia

**Code smell**, also known as **bad smell**, in computer programming code, refers to any symptom in the source code of a program that possibly indicates a deeper problem.[1] According to Martin Fowler,[2] "a code smell is a surface indication that usually corresponds to a deeper problem in the system". Another way to look at smells is with respect to principles and quality:[3] "smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality". Code smells are usually not bugs—they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future. Bad code smells are an important reason for technical debt.[1] Robert C. Martin calls a list of code smells a "value system" for software craftsmanship.[4]

Common code smells

Application-level smells:

- **Duplicated code**: identical or very similar code exists in more than one location.
  - Continued complexity forced usage of very complicated design patterns where simplicity is possible and appropriate.
Dogs vs. Dalmatians

A Dalmatian is-a dog with a spot count. If only we could say that directly...

Inheritance
Code Reuse via Inheritance

The Big Idea

Create new classes from existing classes.

Instead of defining a new class from scratch, explain how it's different from a class you already have:

- additional fields
- additional methods
- changed method bodies

**BUT**

If the old & new classes don't have an is-a relationship, inheritance is probably the wrong tool.
A Dalmatian is-a Dog with a Spot Count

Buddy, Age 5

Zoe, Age 1

47 spots
A Guard Dog is-a Dog that can also Growl

Buddy, Age 5

Rudy, Age 9

public class Dog {
    String myName;
    int myAge;

    public Dog(String n, int a) {
        this.myName = n;
        this.myAge = a;
    }

    public void sayHello() {
        System.out.println("Woof");
    }
}

public class GuardDog extends Dog {
    public GuardDog(String n, int a) {
        super(n,a);
    }

    public void growl() {
        System.out.println("Grrr");
    }
}
A French Poodle is-a Dog that speaks French

Buddy, Age 5

Monty, Age 2

```java
public class Dog {
    String myName;
    int myAge;

    public Dog(String n, int a) {
        this.myName = n;
        this.myAge = a;
    }

    public void sayHello() {
        System.out.println("Woof");
    }
}
```

```java
public class FrenchPoodle extends Dog {
    public FrenchPoodle(String n, int a) {
        super(n,a);
    }

    @Override
    public void sayHello() {
        System.out.println("Ouaf");
    }
}
```
Inheritance Terminology

Dog is a **superclass** and a **supertype**. Dalmatian, GuardDog, and FrenchPoodle are **subclasses** and **subtypes**.

Dalmatian/GuardDog/FrenchPoodle **inherit** from Dog. Dalmatian/GuardDog/FrenchPoodle **extend** Dog.

a Dalmatian **is-a** Dog
a GuardDog **is-a** Dog
a FrenchPoodle **is-a** Dog
Inheritance means a GuardDog is-a Dog!

Dog buddy = new Dog("Buddy", 5);
buddy.sayHello(); Woof

GuardDog rudy = new GuardDog("Rudy", 9);
rudy.sayHello(); Woof
 rudy.growl(); Grrr

Dog arnold = new GuardDog("Arnold", 7);
arold.sayHello(); Woof
Inheritance means a GuardDog is-a Dog!

```java
Dog buddy = new Dog("Buddy", 5);

Dog buddy

GuardDog rudy = new GuardDog("Rudy", 9);

GuardDog rudy

Dog arnold = new GuardDog("Arnold", 7);

Dog arnold
```

The declared type and the actual type can differ.
Fun with is-a

Dog arnold = new Dog("Arnold", 1);
arnold.sayHello(); Woof

arnold = new GuardDog("Arnold", 7);
arnold.sayHello(); Woof

arnold = new Dalmatian("Arnold", 4, 99);
arnold.sayHello(); Woof

arnold = new FrenchPoodle("Arnold", 12);
arnold.sayHello(); Ouaf
Summary

The type checker looks at the **declared type** (not the value) to see if method calls are legal.

\[ \text{x.growl()} \text{ is legal only if the type of x guarantees there's a growl()} \text{ method.} \]

When code runs, Java looks at the **actual object** (not the claimed type) to choose which method to run.

\[ \text{myDog.sayHello()} \text{ does different things, depending on what kind of object myDog is currently referencing.} \]
Inheritance Puzzles

Does this type check in Java?

Dog d = new Dalmatian("Pango", 3, 101);
Inheritance Puzzles

Does this type check in Java?

Dalmatian d = new Dog("Pango", 101);
Inheritance Puzzles

Does this type check in Java?

```java
Dog d = new Dalmatian("Pango", 3, 101);
Dalmatian dm = d;
```
Inheritance Puzzles

Does this type check in Java?

GuardDog gd = new GuardDog("fluffy", 1);
gd.growl();
Does this type check in Java?

```java
Dog d = new GuardDog("fluffy", 1);
d.growl();
```

✘
Inheritance Puzzles

Does this type check in Java?

```java
Dog[] dogs = {new GuardDog("Fluffy", 1),
              new FrenchPoodle("Bob", 7),
              new Dalmatian("Terry", 3, 40)};

for (Dog dog : dogs) {
    dog.sayHello();
}
```

✔

Woof
Ouaf
Woof
Every Class Inherits from Object

is really

```java
class Point {
    @Override
    public String toString() {
        ...
    }
}
```