Does this code compile?

Assuming that `GuardDog` inherits from `Dog`, that its constructor takes a `String` and an `int`, and that it has a `growl` method, does the following code compile? Why or why not?

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

(Your response)
interface
what a piece of code can do

implementation
how a piece of code works

type
describe a set of supported operations

class
implement a type’s operations

subtype
add more operations to an existing type

subclass
re-use/modify an existing implementation

inheritance
usually extends interface and implementation
Declared type

When we declare a variable to be of a particular type, the value of that variable must always be an instance of that type.

If a variable has a type, then the value of that variable can be used anywhere that type is expected.

```java
void f(int x) {
    ...
    value is an int
 ...
}
int value = 3;
f(value);
value is an int
int y = value;

void g(Dog d) {
    ...
    buddy is a Dog
 ...
}
Dog buddy = new Dog(…);
g(buddy);
    buddy is a Dog
Dog myDog = buddy;
```
Subtyping: the “is-a” relationship

Implementing an interface establishes an is-a relationship.

Extending an interface establishes an is-a relationship.

Extending a class establishes an is-a relationship.

If we have the following declaration

```
Type variable;
```

then:

- variable’s declared type is Type.
- variable “is a” Type.
- If Type is an interface, then variable “is” all the interfaces that Type transitively extends.
- If Type is a class, then variable “is” all the classes that Type transitively extends and “is” all the interfaces that Type transitively implements.

Dog buddy;

buddy’s d.t. is Dog

buddy is a Dog

buddy is a Pet and an Animal
Subtyping as substitutability

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type or one of its subtypes.

If a variable has a type, then the value of that variable can be used anywhere that type or one of its supertypes is expected.

```java
void g(Animal A) {
    ...
}

Dog buddy = new Dog(...);
g(buddy);

Animal myDog = buddy;
```
Subtyping as substitutability

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type or one of its subtypes.

If a variable has a type, then the value of that variable can be used anywhere that type or one of its supertypes is expected.

```java
void g(Animal A) {
    ...
}

Dog buddy = new Dog(...);
g(buddy);
Animal myDog = buddy;
```
Subtyping as substitutability

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type or one of its subtypes.

If a variable has a type, then the value of that variable can be used anywhere that type or one of its supertypes is expected.

```java
void g(Animal A) {
    ...
}
...

Dog buddy = new Dog(...);
g(buddy);
```

`is-a`
Declared type vs actual type

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

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

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

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

Does this type check in Java?

Cat c = new Cat("Nala", 14);
Inheritance Puzzles

Does this type check in Java?

```
Cat c = new Cat("Nala", 14);
c.speak();
```

✔

Meow
Inheritance Puzzles

Does this type check in Java?

Animal a = new Cat("Nala", 14);
Inheritance Puzzles

Does this type check in Java?

Animal a = new Cat("Nala", 14);
a.speak();
Does this type check in Java?

```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?

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();
Inheritance Puzzles

Does this type check in Java?

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

✘
Courtesy of Prof. Bassman
Graphs

nope

nope

yep!
The Seven Bridges of Königsberg

Can you:
- start at point A,
- cross every bridge only once,
- and return to point A?

Leonard Euler

No

Koenigsberg, Map by Merian-Erben 1652
 commons.wikimedia.org/wiki/File:Image-Koenigsberg_Map_by_Merian-Erben_1652.jpg

commons.wikimedia.org/wiki/File:Leonhard_Euler.jpg
We need a model
A set of **nodes/vertices** (places), and a set of **edges** (links)
Graphs represent relationships

Like what?
A node is …
An edge is…
Graphs represent relationships

Like Facebook

A node is a Facebook user
An edge is a “friendship”
Undirected graph

We can “traverse the edge” in both directions.
The relationship is “mutual”.

← important vocabulary!
Undirected graph

We can “traverse the edge” in both directions.
The relationship is “mutual”.

← important vocabulary!
Graphs represent relationships

Like Twitter
A node is a Twitter user
An edge is a “follow”
Directed graph

We can “traverse the edge” in one direction.
The relationship is “one way”.

← important vocabulary!
Directed graph

We can “traverse the edge” in one direction.
The relationship is “one way”.

edge "source"  edge "destination"
A → B
A → B
A → D
B → C
B → C
B → D
D → C
Graphs represent relationships

Like highways
A node is a city
An edge is a highway from one city to another
Weighted graph

Information (usually “cost”) associated with each edge
Graphs represent relationships

Like flights

A node is a city
An edge is a flight from one city to another
Directed weighted graph

Information (usually “cost”) associated with each edge

<table>
<thead>
<tr>
<th>edge &quot;source&quot;</th>
<th>edge &quot;weight&quot;</th>
<th>edge &quot;destination&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>100</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>85</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>90</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>90</td>
<td>C</td>
</tr>
</tbody>
</table>

→ important vocabulary!
<table>
<thead>
<tr>
<th></th>
<th>Undirected</th>
<th>Directed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unweighted</td>
<td><img src="image1" alt="facebook" /></td>
<td><img src="image2" alt="twitter" /></td>
</tr>
<tr>
<td>Weighted</td>
<td><img src="image3" alt="highways" /></td>
<td><img src="image4" alt="flights" /></td>
</tr>
</tbody>
</table>

weights = distance, time, cost
D has one adjacent edge. C is adjacent to D.

There are two paths from A to D.

C is reachable from A.
Complete graph

There is an edge between each pair of nodes.

In other words, each node is adjacent to every other node.

To be true in a directed graph, the edges must go in both directions.
Connected graph

There is a path between each pair of nodes.

In other words, each node is reachable from every other node.

*If this is true in a directed graph, the graph is “strongly connected”.*
a **sparse** graph has few edges

a **dense** graph has many edges

重要的词汇库!
There may be an infinite number of paths to a node.

Acyclic: there are a finite number of paths to a node.

Cyclic: there may be an infinite number of paths to a node.
We’ve seen graphs before!

- A linked list is a graph; a tree is a graph.

**Linked List**
- A connected, directed, acyclic graph.

**Tree**
- A connected, directed, acyclic graph.