Check07C

Number of students enrolled in the course: 198

Number of students that eChecked Check07C: 18 (9%)

Check08A

Number of students enrolled in the course: 191

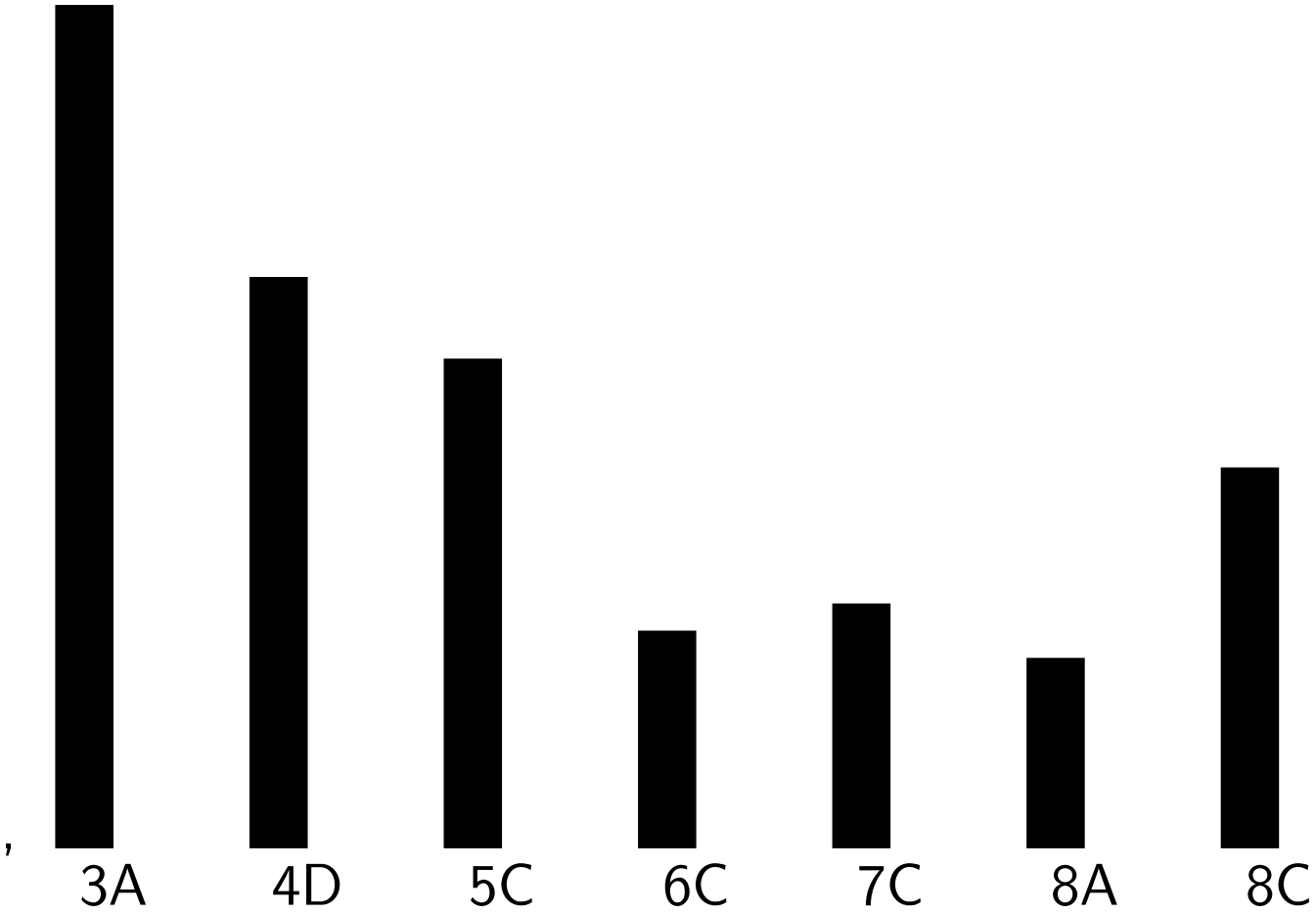
Number of students that eChecked Check08A: 14 (7%)

Check08C

Number of students enrolled in the course: 173

Number of students that eChecked Check08C: 24 (14%)

Checks



Big-O Notation

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ be functions. Then $f \in O(g)$ if

$$\exists M \in \mathbb{N} : \exists F \in \mathbb{N} : \forall n \geq M : f(n) \leq F \times g(n)$$

A note about the big-O notation can be found at http://www.cse.yorku.ca/course/1020/sectionA/complexity.pdf.

Inheritance

Definition

Inheritance is a binary relation on classes. The pair (C, P) of classes is in the inheritance relation if the API of the class C (child) contains

```
class C extends P
```

The API of the class P (parent) may (but does not have to) contain

```
Direct Known Subclasses: C
```

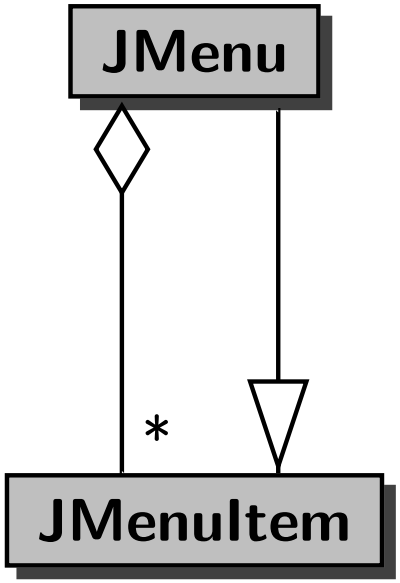
The inheritance relation is also known as the is-a relation. Instead of saying that (C, P) is in the inheritance relation, we often simply say that C is-a P.

Inheritance

We restrict ourselves to well-designed classes and the client perspective.

- Constructors are not inherited from the superclass.
- Each public non-static (final) attribute is inherited from the superclass.
- Each public non-static method is inherited from the superclass. If the subclass defines a method with the same signature (name and list of types of parameters), then the inherited method is overridden in the subclass.

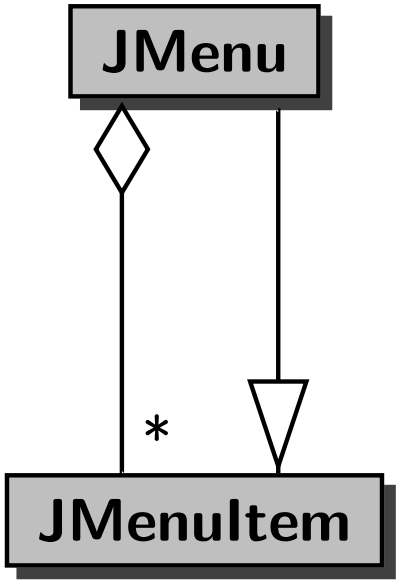
Question



Question

What do the relations capture?

Question



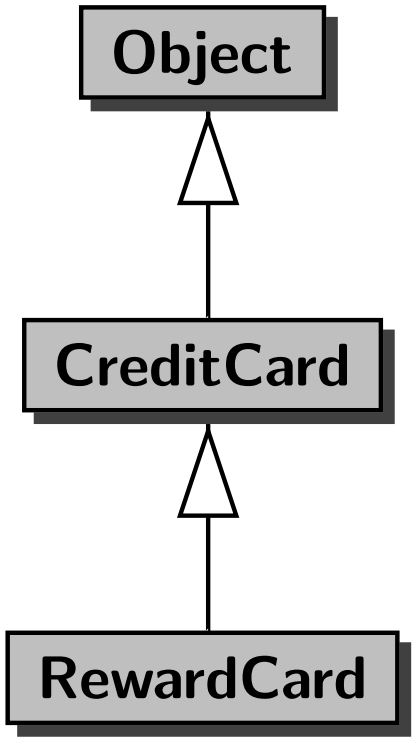
Question

What do the relations capture?

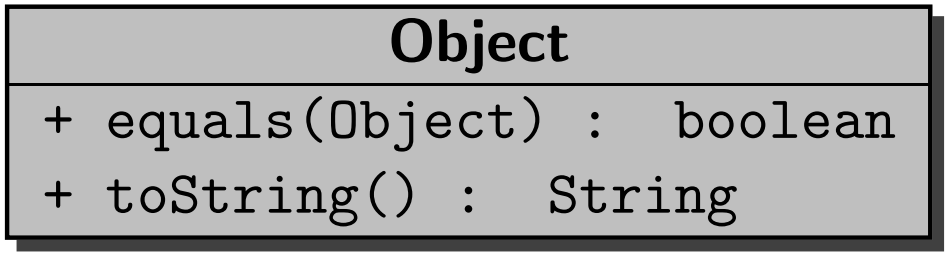
Answer

A menu can have multiple items and a (nested) menu is an item.

CreditCard, RewardCard and Object



The Object Class



If `object` is an `Object` and not `null`, then `object.equals(other)` is equivalent to `object == other`.

If `object` is an `Object` and not `null`, then `object.toString()` returns a String such as `java.lang.Object@3e25a5` where the hexadecimal `3e25a5` represents usually the memory address of `object`.

The Method equals

Question

Each class has an `equals(Object)` method. Why?

The Method equals

Answer

Obviously, the `Object` class has an `equals(Object)` method.

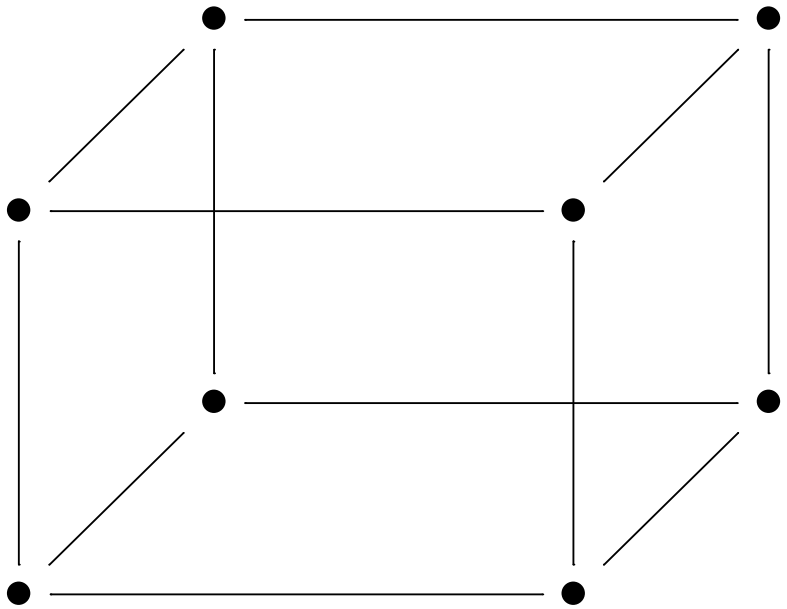
Let C be a class different from the `Object` class. Since each class is a descendent of the `Object` class , we know that there exist classes $C_1, \ldots, C_n$ such that C_1 extends `Object`, C_2 extends C_1, $\ldots$, C_n extends C_{n-1} and C extends C_n. Since `Object` has a public `equals(Object)` method and C_1 extends `Object`, C_1 has a public `equals(Object)` method (inherits it but may override it). Since C_1 has a public `equals(Object)` method and C_2 extends C_2, C_2 has a public `equals(Object)` method (inherits it but may override it). $\ldots$ Since C_n has a public `equals(Object)` method and C extends C_n, C has a public `equals(Object)` method (inherits it but may override it).

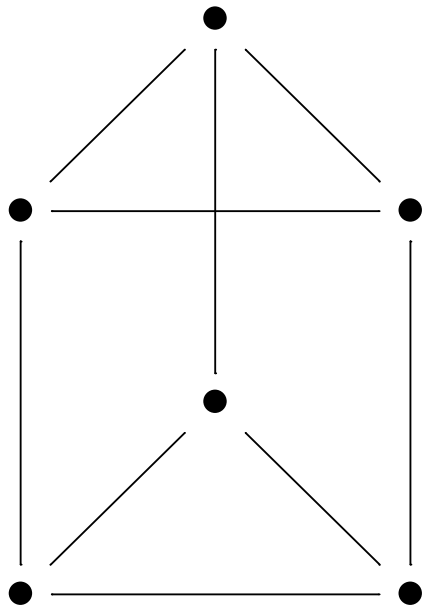
The Method `toString`

Property

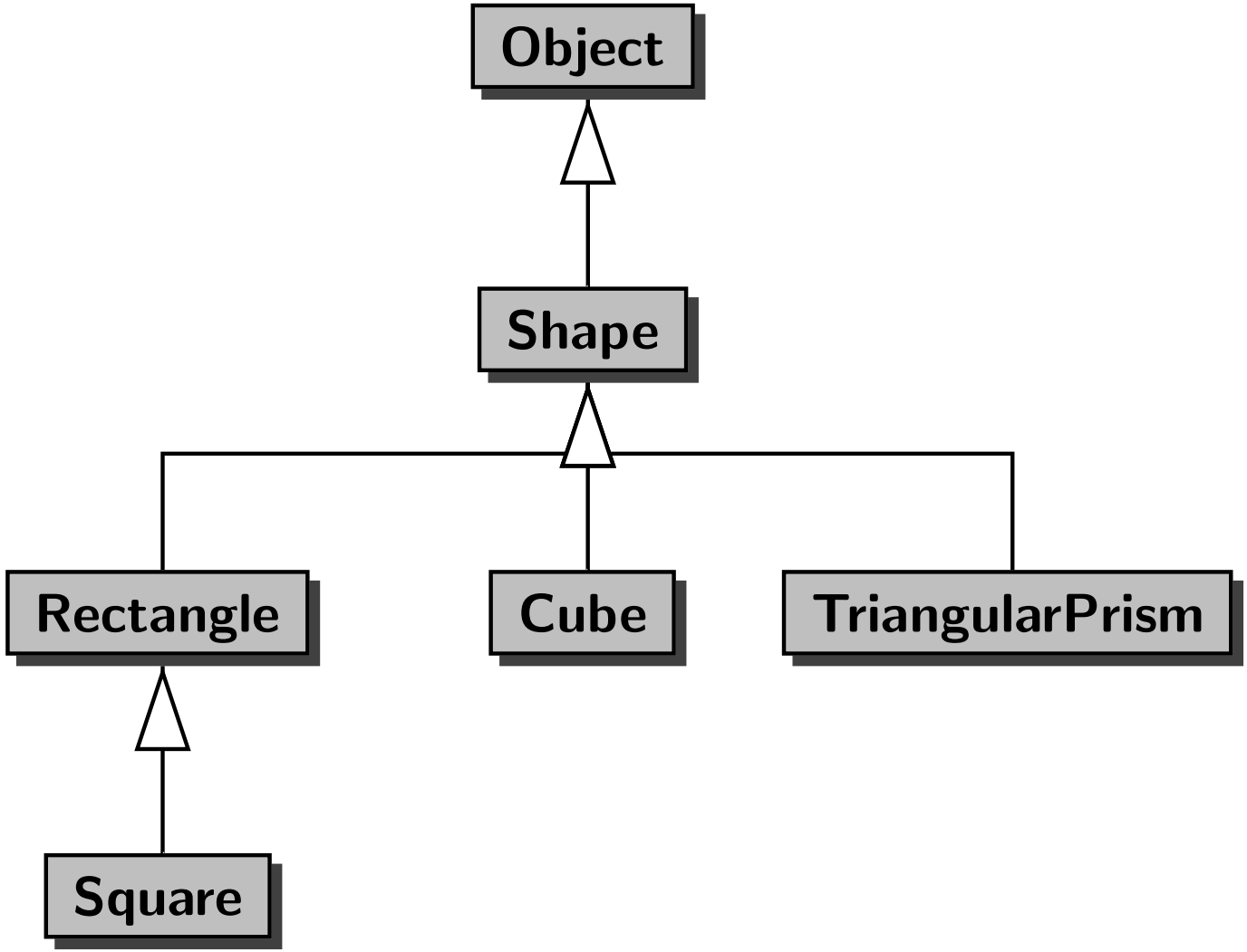
Each class has an `toString()` method.

Shapes





Shapes



Random Shapes

Question

The classes `Rectangle`, `Cube`, `TriangularPrism`, and `Square` each have a static method `getRandom()`. Create a random shape.

Random Shapes

```
Random random = new Random();
final int NUMBER_OF_SHAPES = 4;
... shape;
switch (random.nextInt(NUMBER_OF_SHAPES))
{
   case 0 : shape = Rectangle.getRandom(); break;
   case 1 : shape = Cube.getRandom(); break;
   case 2 : shape = TriangularPrism.getRandom(); break;
   case 3 : shape = Square.getRandom(); break;
   default : shape = null; break;
}
```

Random Shapes

```
Random random = new Random();
final int NUMBER_OF_SHAPES = 4;
... shape;
switch (random.nextInt(NUMBER_OF_SHAPES))
{
   case 0 : shape = Rectangle.getRandom(); break;
   case 1 : shape = Cube.getRandom(); break;
   case 2 : shape = TriangularPrism.getRandom(); break;
   case 3 : shape = Square.getRandom(); break;
   default : shape = null; break;
}
```

What is the type of `shape`?

Random Shapes

```
Shape shape = Rectangle.getRandom();
```

Question

What is the return type of the method `getRandom`?

Random Shapes

```
Shape shape = Rectangle.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Rectangle`.

Random Shapes

```
Shape shape = Rectangle.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Rectangle`.

Question

Why can we assign a `Rectangle` object to a `Shape` variable?

Random Shapes

```
Shape shape = Rectangle.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Rectangle`.

Question

Why can we assign a `Rectangle` object to a `Shape` variable?

Answer

Because a `Rectangle` is-a `Shape`.

Random Shapes

```
Shape shape = Cube.getRandom();
```

Question

What is the return type of the method `getRandom`?

Random Shapes

```
Shape shape = Cube.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Cube`.

Random Shapes

```
Shape shape = Cube.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Cube`.

Question

Why can we assign a `Cube` object to a `Shape` variable?

Random Shapes

```
Shape shape = Cube.getRandom();
```

Question

What is the return type of the method `getRandom`?

Answer

`Cube`.

Question

Why can we assign a `Cube` object to a `Shape` variable?

Answer

Because a `Cube` is-a `Shape`.

Types

```
Shape shape = Rectangle.getRandom();
```

Question

What is the declared type of `shape`?

Types

```
Shape shape = Rectangle.getRandom();
```

Question

What is the declared type of `shape`?

Answer

`Shape`.

Types

```
Shape shape = Rectangle.getRandom();
```

Question

What is the declared type of `shape`?

Answer

`Shape`.

Question

What is the actual type of `shape`?

Types

```
Shape shape = Rectangle.getRandom();
```

Question

What is the declared type of `shape`?

Answer

`Shape`.

Question

What is the actual type of `shape`?

Answer

`Rectangle`.

The Substitutability Principle

This principle is described in the textbook as

When a parent is expected, a child is accepted.

Barbara Liskov and Jeannette Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, November 1994.

Barbara Liskov

Barbara Liskov is the Ford Professor of Engineering in the MIT School of Engineering's Electrical Engineering and Computer Science department and an Institute Professor at the Massachusetts Institute of Technology. She earned her BA in mathematics at the University of California, Berkeley in 1961. In 1968 Stanford University made her one of the first women in the United States to be awarded a Ph.D. from a computer science department. Liskov won the Turing Award in 2008.

source: Mirko Raner

Jeannette Wing

Jeannette Wing is the President's Professor of Computer Science at Carnegie Mellon University. Wing earned her S.B. and S.M. in Electrical Engineering and Computer Science at MIT in 1979. In 1983, she earned her Ph.D. in Computer Science at MIT.

source: www.cmu.edu

Comparing Shapes

```
Rectangle rectangle = Rectangle.getRandom();
Square square = Square.getRandom();
output.println(rectangle.compareTo(square));
```

Question

What is the parameter type of the method `compareTo`?

Comparing Shapes

```
Rectangle rectangle = Rectangle.getRandom();
Square square = Square.getRandom();
output.println(rectangle.compareTo(square));
```

Question

What is the parameter type of the method `compareTo`?

Answer

`Rectangle`.

Comparing Shapes

```
Rectangle rectangle = Rectangle.getRandom();
Square square = Square.getRandom();
output.println(rectangle.compareTo(square));
```

Question

What is the parameter type of the method `compareTo`?

Answer

`Rectangle`.

Question

Why can we provide a `Square` object as an argument?

Comparing Shapes

```
Rectangle rectangle = Rectangle.getRandom();
Square square = Square.getRandom();
output.println(rectangle.compareTo(square));
```

Question

What is the parameter type of the method `compareTo`?

Answer

`Rectangle`.

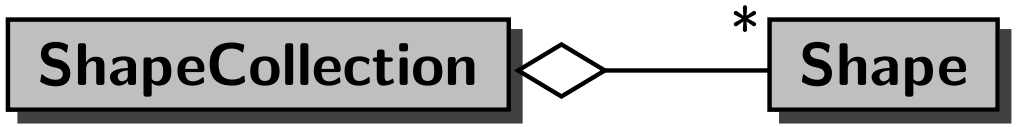
Question

Why can we provide a `Square` object as an argument?

Answer

Because a `Square` is-a `Rectangle`.

A Collection of Shapes



Question

Create a random collection of shapes and print its shapes each on a seperate line.

A Collection of Shapes

```
ShapeCollection collection = ShapeCollection.getRandom();
for (Shape shape : collection)
{
   output.println(shape.toString());
}
```

Early (Static) Binding

`object.method(argument`$_1$`, ... argument`$_n$`);`

- Determine the declared type of `object`: C.
- Determine the declared types of `argument`$_1$, ..., `argument`$_n$: $C_1, \ldots, C_n$.
- Of all methods named `method` in class C, pick the one whose parameter types match $(C_1, \ldots C_n)$ best: `method`$(T_1, \ldots T_n)$ of C.
  (If no match can be found, the compiler issues an error.)

Late (Dynamic) Binding

`object.method(argument`$_1$`, ... argument`$_n$`);`

Assume that early binding resulted in `method(`$T_1, \ldots T_n$`)` of C.

- Determine the actual type of `object`: T.

The late binding results in `method(`$T_1, \ldots T_n$`)` of T.

Note that the signature does not change during the late binding (for efficiency reasons).

Early and Late Binding

Property

If early binding results in `method`$(T_1, \ldots T_n)$ of C then each subclass of C has a method with signature `method`$(T_1, \ldots T_n)$.

Proof

Similar to the proof that each class has a `toString()` method.

Corollary

Late binding never fails.

Early Binding

```
for (Shape shape : collection)
{
    output.println(shape.toString());
}
```

Question

What is the early binding for

`shape.toString()`

Early Binding

```
for (Shape shape : collection)
{
    output.println(shape.toString());
}
```

Question

What is the early binding for

`shape.toString()`

Answer

`toString()` of `Shape` (inherited from `Object`).

Late Binding

```
for (Shape shape : collection)
{
    output.println(shape.toString());
}
```

Question

What is the late binding for

`shape.toString()`

Late Binding

```
for (Shape shape : collection)
{
    output.println(shape.toString());
}
```

Question

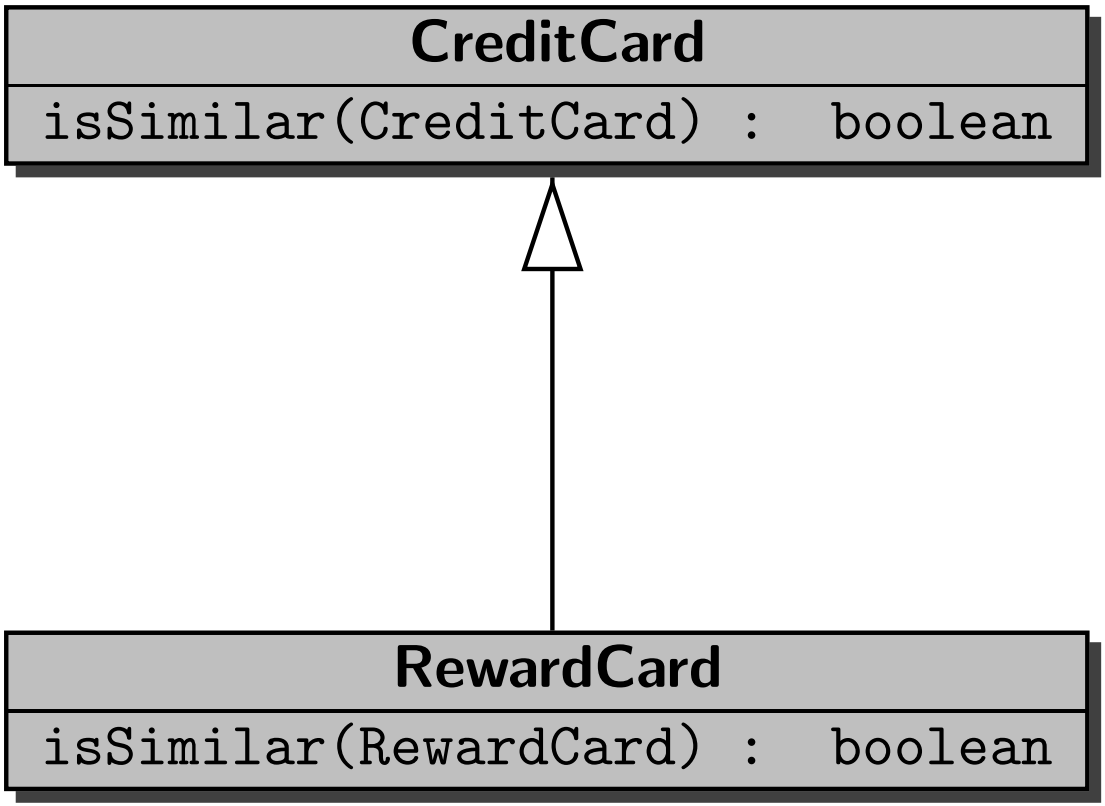
What is the late binding for

`shape.toString()`

Answer

`toString()` of `Rectangle`, `Cube`, `TriangularPrism` or `Square`.

Contrived Example



Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
one.isSimilar(two);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

`one.isSimilar(two);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
one.isSimilar(two);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
one.isSimilar(two);
```

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
one.isSimilar(three);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
one.isSimilar(three);
```

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
one.isSimilar(three);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

`one.isSimilar(three);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar(one);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar(one);
```

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
three.isSimilar(one);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

`three.isSimilar(one);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar(one);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar(one);
```

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
three.isSimilar(one);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

`three.isSimilar(one);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar(four);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

`three.isSimilar(four);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
three.isSimilar(four);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

`three.isSimilar(four);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
four.isSimilar(three);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

`four.isSimilar(three);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
four.isSimilar(three);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

`four.isSimilar(three);`

Answer

`isSimilar(CreditCard)` of `CreditCard`.

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar((RewardCard) four);
```

Early Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the early binding for

```
three.isSimilar((RewardCard) four);
```

Answer

`isSimilar(RewardCard)` of `RewardCard`.

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
three.isSimilar((RewardCard) four);
```

Late Binding

```
CreditCard one = new CreditCard(...);
CreditCard two = new CreditCard(...);
RewardCard three = new RewardCard(...);
CreditCard four = new RewardCard(...);
```

Question

What is the late binding for

```
three.isSimilar((RewardCard) four);
```

Answer

`isSimilar(RewardCard)` of `RewardCard`.