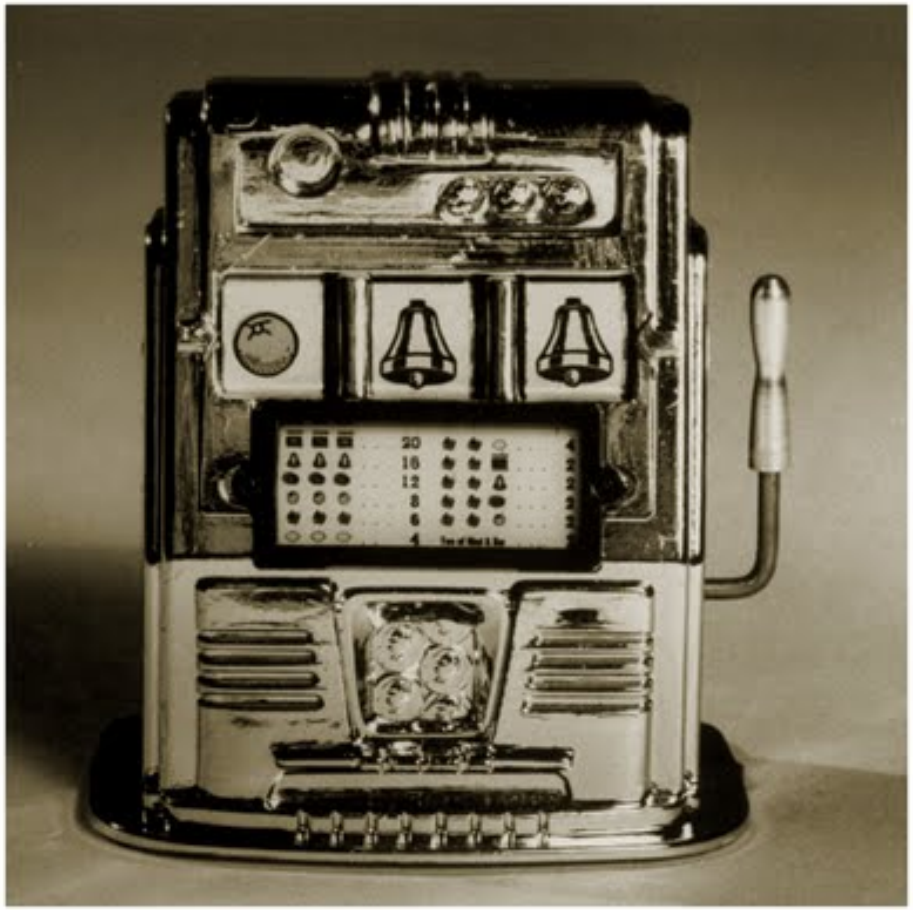
One Armed Bandit



source: http://dogbeforewicket.blogspot.ca

One Armed Bandit Utility

```
/**
 * Returns the winnings from one pull of the one armed
 * bandit.
 *
 * @param coin the coin deposited in the one armed bandit.
 * @return the payoff from one pull of the lever.
 */
public static List<Coin> pull(Coin coin)
```

Casino App

```
List<Coin> wallet = new ArrayList<Coin>();
wallet.add(new Coin());

final int TRIES = 10;
int tries = 0;

while (tries < TRIES && !wallet.isEmpty())
{
  Coin coin = wallet.remove(0);
  List<Coin> winnings = OneArmedBandit.pull(coin);
  wallet.addAll(winnings);
  tries++;
}

System.out.printf("After %d tries, %d coins left%n",
                  tries, wallet.size());
```

The `Coin` Class

Problem

Implement the `Coin` class. We only need to be able to create `Coin` objects, using the default constructor. We do not need any methods of the `Coin` class.

The `Coin` Class

Problem

Implement the `Coin` class. We only need to be able to create `Coin` objects, using the default constructor. We do not need any methods of the `Coin` class.

Now that we have the `Coin` class, we can run the Casino app.

How Many Coins?

Question

You win 37 coins in 10 tries. How many `Coin` objects are there stored in memory?

How Many Coins?

Question

You win 37 coins in 10 tries. How many `Coin` objects are there stored in memory?

Answer

At least 37.

How Many Coins?

Question

You win 37 coins in 10 tries. How many `Coin` objects are there stored in memory?

Answer

At least 37.

Question

Instead of storing `Coin` objects in your wallet, could you store aliases of a single `Coin` object in your wallet?

How Many Coins?

Question

You win 37 coins in 10 tries. How many `Coin` objects are there stored in memory?

Answer

At least 37.

Question

Instead of storing `Coin` objects in your wallet, could you store aliases of a single `Coin` object in your wallet?

Answer

Yes.

A Single Coin

Problem

Modify the `Coin` class so that the client can create at most one `Coin` object.

A Single Coin

Question

Can the client use a constructor to create the `Coin` object?

A Single Coin

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

A Single Coin

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

Question

Why not?

A Single Coin

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

Question

Why not?

Answer

If we provide a public constructor, clients can invoke it as many times as they want and, hence, create as many `Coin` objects as they want.

A Single Coin

Question

Since the client cannot use a constructor, what other options does the client have?

A Single Coin

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

A Single Coin

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

Question

Can the method be non-static?

A Single Coin

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

Question

Can the method be non-static?

Answer

No, because you would need a `Coin` object to invoke it on (and we are trying to create a `Coin` object). So the method is static.

A Single Coin

Question

What is the return type of this static method?

A Single Coin

Question

What is the return type of this static method?

Answer

`Coin`.

A Single Coin

Question

What is the return type of this static method?

Answer

`Coin`.

Question

Does it have any parameters?

A Single Coin

Question

What is the return type of this static method?

Answer

`Coin`.

Question

Does it have any parameters?

Answer

No.

A Single Coin

```
public static Coin getInstance()
```

Question

The method name suggests that the `Coin` class has a static attribute. What is its name and type?

A Single Coin

```
public static Coin getInstance()
```

Question

The method name suggests that the `Coin` class has a static attribute. What is its name and type?

Answer

`instance` and `Coin`.

A Single Coin

Question

Where do we initialize the attribute?

A Single Coin

Question

Where do we initialize the attribute?

Answer

In the declaration.

A Single Coin

Question

Where do we initialize the attribute?

Answer

In the declaration.

Question

How?

A Single Coin

Question

Where do we initialize the attribute?

Answer

In the declaration.

Question

How?

Answer

private static Coin instance = new Coin();

A Single Coin

Question

But as we argued earlier, we cannot provide a public constructor. So, how can we use `new Coin()` in the `Coin` class?

A Single Coin

Question

But as we argued earlier, we cannot provide a public constructor. So, how can we use `new Coin()` in the `Coin` class?

Answer

By adding a private default constructor.

A Single Coin

Question

But as we argued earlier, we cannot provide a public constructor. So, how can we use `new Coin()` in the `Coin` class?

Answer

By adding a private default constructor.

Question

Now that we have declared and initialized the `instance` attribute, how do we implement the `getInstance` method?

A Single Coin

Question

But as we argued earlier, we cannot provide a public constructor. So, how can we use `new Coin()` in the `Coin` class?

Answer

By adding a private default constructor.

Question

Now that we have declared and initialized the `instance` attribute, how do we implement the `getInstance` method?

Answer

```
return Coin.instance;
```

Singleton Design Pattern

The pattern to ensure that at most one object of a particular class can be created is known as the singleton design pattern.

The example we presented is contrived. In case object represent physical entities, such as a connection to a database, the singleton design pattern comes in handy.

Coin with Values

Problem

Modify the `Coin` class some that each coin has a value (of type int). Make the class immutable.

Coin with Values

Problem

Modify the `Coin` class so that the client can create at most one `Coin` object for each value. That is, the client can create different `Coin` objects but only if they all have different values.

Coin with Values

Question

Can the client use a constructor to create the `Coin` object?

Coin with Values

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

Coin with Values

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

Question

Why not?

Coin with Values

Question

Can the client use a constructor to create the `Coin` object?

Answer

No.

Question

Why not?

Answer

If we provide a public constructor, clients can invoke it as many times as they want and, hence, create as many `Coin` objects with the same value as they want.

Coin with Values

Question

Since the client cannot use a constructor, what other options does the client have?

Coin with Values

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

Coin with Values

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

Question

Can the method be non-static?

Coin with Values

Question

Since the client cannot use a constructor, what other options does the client have?

Answer

Methods.

Question

Can the method be non-static?

Answer

No, because you would need a `Coin` object to invoke it on (and we are trying to create a `Coin` object). So the method is static.

Coin with Values

Question

What is the return type of this static method?

Coin with Values

Question

What is the return type of this static method?

Answer

`Coin`.

Coin with Values

Question

What is the return type of this static method?

Answer

`Coin`.

Question

Does it have any parameters?

Coin with Values

Question

What is the return type of this static method?

Answer

`Coin`.

Question

Does it have any parameters?

Answer

Yes, the value of the coin.

Coin with Values

```
public static Coin getInstance(int value)
```

Coin with Values

```
public static Coin getInstance(int value)
```

Question

For each value, we have to store a `Coin`. How can we do that?

Coin with Values

```
public static Coin getInstance(int value)
```

Question

For each value, we have to store a `Coin`. How can we do that?

Answer

```
private static Map<Integer, coin> instance
```

Coin with Values

Question

How do we initialize the attribute?

Coin with Values

Question

How do we initialize the attribute?

Answer

```
private static Map<Integer, coin> instance
    = new HashMap<Integer, Coin>();
```

Coin with Values

Question

How do we implement the `getInstance` method?

Coin with Values

Question

How do we implement the `getInstance` method?

Answer

```
if (!instance.containsKey(value))
{
   Coin.instance.put(value, new Coin(value));
}
return Coin.instance.get(value);
```

Multiton Design Pattern

The pattern to ensure that at most one object with a particular state can be created is known as the multiton design pattern.

Immutable classes such as String and Integer implement this design pattern.

Concurrency

"Doing multiple things at the same time"

Question

What happens when we execute

```
public static Coin getInstance()
{
  if (Coin.instance == null)
  {
    Coin.instance = new Coin();
  }
  return Coin.instance;
}
```

twice but at the same time?

Java Bytecode

```
getstatic Coin.instance
// get the value of the static attribute
ifnonnull 6
// if the value is not null, go to line 6
new Coin
// create a new Coin object
...
putstatic Coin.instance
// set the value of the static attribute
getstatic Coin.instance
// get the value of the static attribute
areturn
```