CSCI 200: Foundational Programming Concepts & Design Lecture 30

Object-Oriented Programming & Inheritance: Abstract Classes & Interfaces

SOLID Principles

Virtual Classes

- Classes with virtual functions need a virtual destructor
  - When deleting pointer, need to delete subtype object
- Explicitly declare parent destructor as virtual
  - Typically don’t mark child destructor as override (names don’t match)

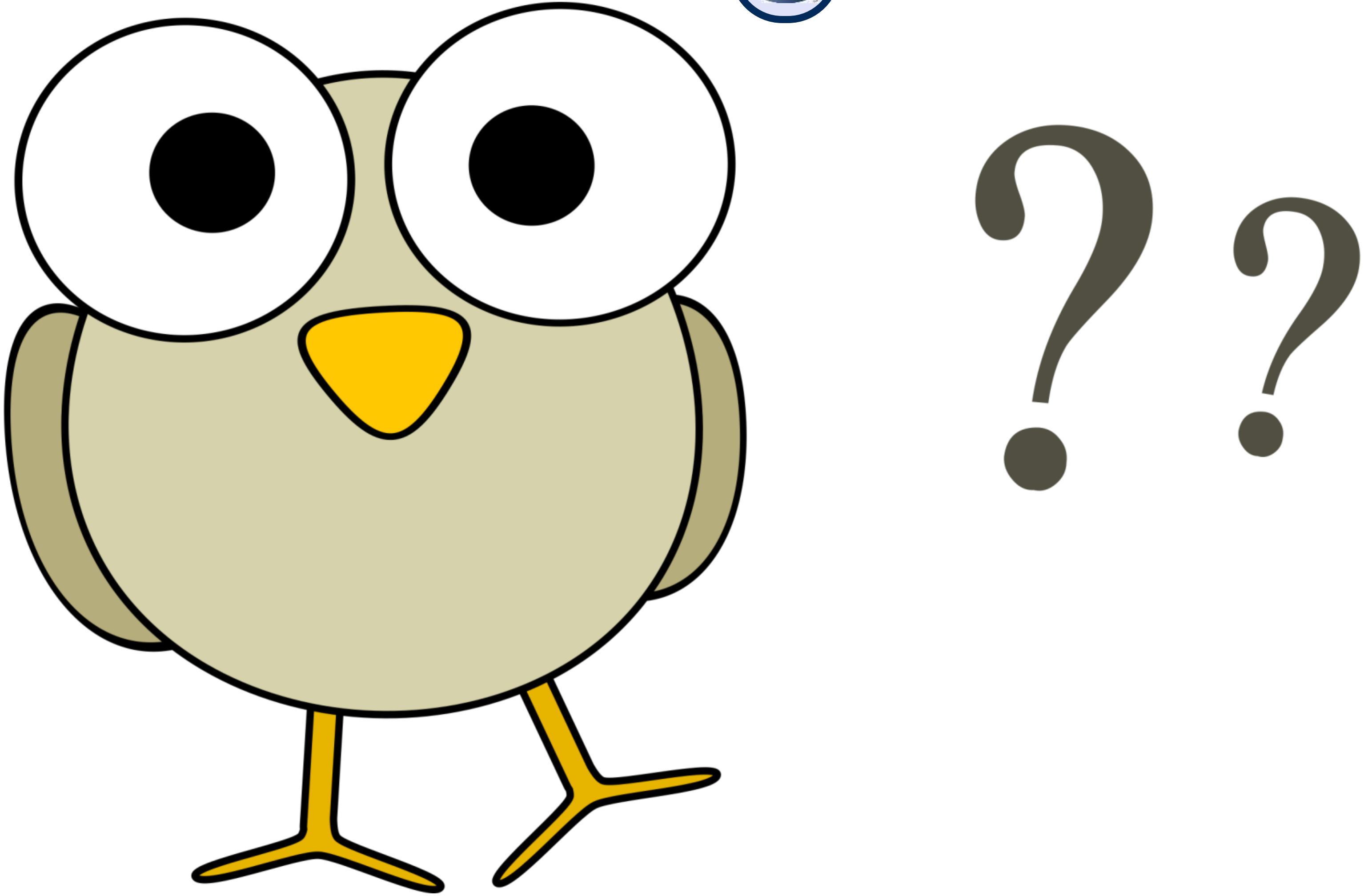
```
class Animal {
public:
  virtual ~Animal() { cout << "Destroying an animal" << endl; }
  virtual void speak() const { cout << "..." << endl; }
};

class Dog : public Animal {
public:
  ~Dog() { cout << "Destroying a dog" << endl; }
  void speak() const override { cout << "bark" << endl; }
};
```

Previously in CSCI 200

- Runtime Polymorphism
  - Virtual function implementations bound at run time based on pointer object type

Questions?



Learning Outcomes For Today

- Give examples of polymorphism at run-time through subtype polymorphism with virtual functions.
- Define abstract classes and discuss their limitations.
- Define interface.
- Define the SOLID Principles.
- Discuss the Interface Segregation Principle.

On Tap For Today

- Abstract Classes
- SOLID Principles
- Practice

On Tap For Today

- Abstract Classes
- SOLID Principles
- Practice

Virtual Functions

```
class Animal {
public:
  virtual ~Animal() {}
  virtual void speak() const  { cout << "..." << endl; }   // base implementation
};
class Dog : public Animal {
public:
  void speak() const override { cout << "bark" << endl; }  // override base
};
class Cat : public Animal {
public:
  void speak() const override { cout << "meow" << endl; }  // override base
};
```

Pure Virtual Functions

- Virtual Function with no default implementation
  - Pure Virtual Function == Abstract Function

```
class Animal {
public:
  virtual ~Animal() {}
  virtual void speak() const = 0;                         // abstract declaration
};
class Dog : public Animal {
public:
  void speak() const override { cout << "bark" << endl; } // concrete definition
};
class Cat : public Animal {
public:
  void speak() const override { cout << "meow" << endl; } // concrete definition
};
```

Abstract Classes

- Class with at least one abstract function is an Abstract Class
  - Cannot instantiate Abstract Classes

```
Animal mythicalAnimal;           // Error!! – Animal is abstract
mythicalAnimal.speak();          // Error!! – speak undefined

Dog odie;                        // ok – Dog is concrete
Cat garfield;                    // ok – Cat is concrete

Animal* pGarfieldAndFriends;     // pointer to an Animal object
pGarfieldAndFriends = &odie;     // ok – Dog is an Animal
pGarfieldAndFriends->speak();    // resolves to Dog::speak()
pGarfieldAndFriends = &garfield; // ok – Cat is an Animal
pGarfieldAndFriends->speak();    // resolves to Cat::speak()
// can only ever point at concrete things
```

Abstract Class

- Class with at least one abstract function
  - And
    - Data members to track state
    - OR Non-abstract functions

```cpp
// Animal is an abstract class
// cannot instantiate it

class Animal {
public:
  virtual ~Animal() {} // classes with virtual functions need a virtual destructor
  virtual void speak() const = 0;                // abstract declaration
  std::string getName() const { return mName; }  // concrete definition
  void setName(const std::string NEW_NAME) { mName = NEW_NAME; }

protected:
  std::string mName;                             // concrete data member
};
```

Interfaces

- Abstract Class with ONLY abstract functions
  - Declares what should be done,
    doesn't define how it should be done

```
// IList is an interface
// cannot instantiate it
template<typename T>
class IList {
public:
  virtual ~IList() {}            // C++ requires a virtual destructor be present
  virtual void pushFront(T) = 0;
  virtual void pushBack(T) = 0;
  virtual T popFront() = 0;
  virtual T popBack() = 0;
  virtual void insert(int, T) = 0;
  virtual T remove(int) = 0;
  virtual unsigned int size() const = 0;
  virtual T& at(int) = 0;
  virtual void set(int, T) = 0;
  // ...
};
```

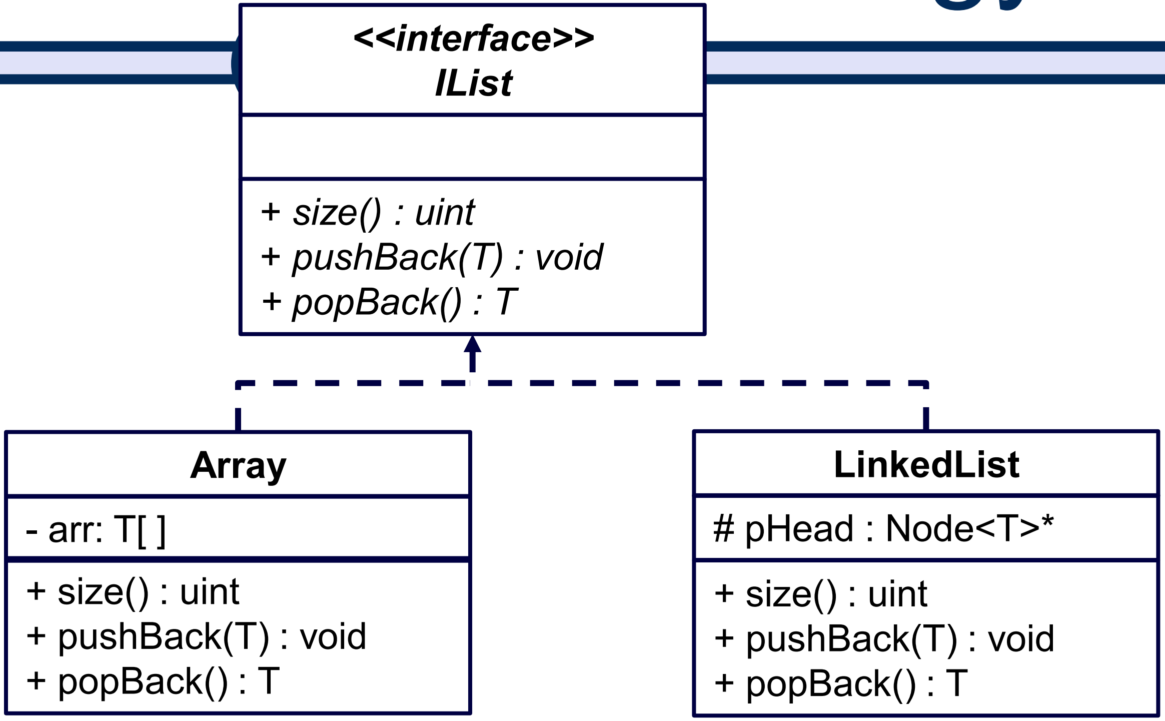
Interfaces

- Abstract Class with ONLY abstract functions
  - Declares what should be done, doesn’t define how it should be done

```
// IList is an interface
// cannot instantiate it
template<typename T>
class IList {
public:
  virtual ~IList() = default;    // C++ requires a virtual destructor be present
  virtual void pushFront(T) = 0;
  virtual void pushBack(T) = 0;
  virtual T popFront() = 0;
  virtual T popBack() = 0;
  virtual void insert(int, T) = 0;
  virtual T remove(int) = 0;
  virtual unsigned int size() const = 0;
  virtual T& at(int) = 0;
  virtual void set(int, T) = 0;
  // ...
};
```

UML Notation & Terminology

- **ClassName**
- + public
- # protected
- - private
- ↑ extends
- ⇡ implements
- *abstract*



<<interface>>
IList

+ size() : uint
+ pushBack(T) : void
+ popBack() : T

Array

- arr: T[ ]

+ size() : uint
+ pushBack(T) : void
+ popBack() : T

LinkedList

# pHead : Node<T>*

+ size() : uint
+ pushBack(T) : void
+ popBack() : T

Design Principle

- “Program to an interface, not an implementation”
- Leverage polymorphism
  - Rely only on what operations can be done
  - More maintainable
  - Can change behavior at run time

Program to an Interface

```
class ISpeaker {
public:
  virtual ~ISpeaker() {}
  virtual void sayHello() = 0;
  virtual void askHowAreYou() = 0;
};

class EnglishSpeaker : public ISpeaker {
public:
  void sayHello() { cout << "Hello" << endl; }
  void askHowAreYou() { cout << "How are you?" << endl; }
};

class ItalianSpeaker : public ISpeaker {
public:
  void sayHello() { cout << "Ciao" << endl; }
  void askHowAreYou() { cout << "Come stai?" << endl; }
};

int main() {
  ISpeaker *pSpeaker = get_speaker(); // returns a concrete speaker object
  pSpeaker->sayHello();
  pSpeaker->askHowAreYou();
}
```

On Tap For Today

- Abstract Classes
- SOLID Principles
- Practice

SOLID Principles

- Set of design principles for object-oriented software development

- S – Single Responsibility Principle
- O – Open/Closed Principle
- L – Liskov Substitution Principle
- I – Interface Segregation Principle
- D – Dependency Inversion

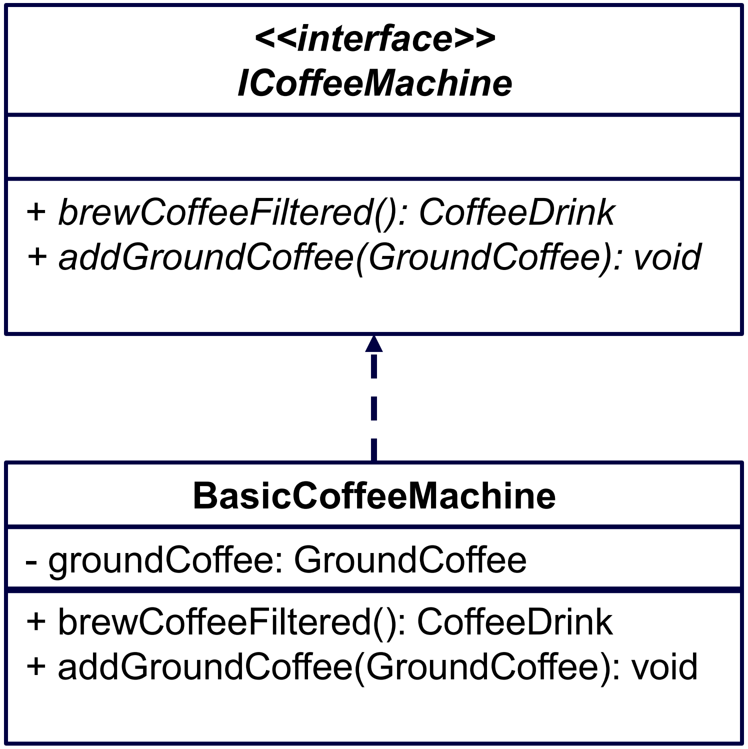
On Tap For Today

- Abstract Classes
- SOLID Principles
- Practice

Interface Segregation Principle

- "*Clients should not be forced to depend upon interfaces that they do not use.*"
  - Robert C. Martin when consulting for Xerox

Interface Segregation Principle



<<interface>>
ICoffeeMachine

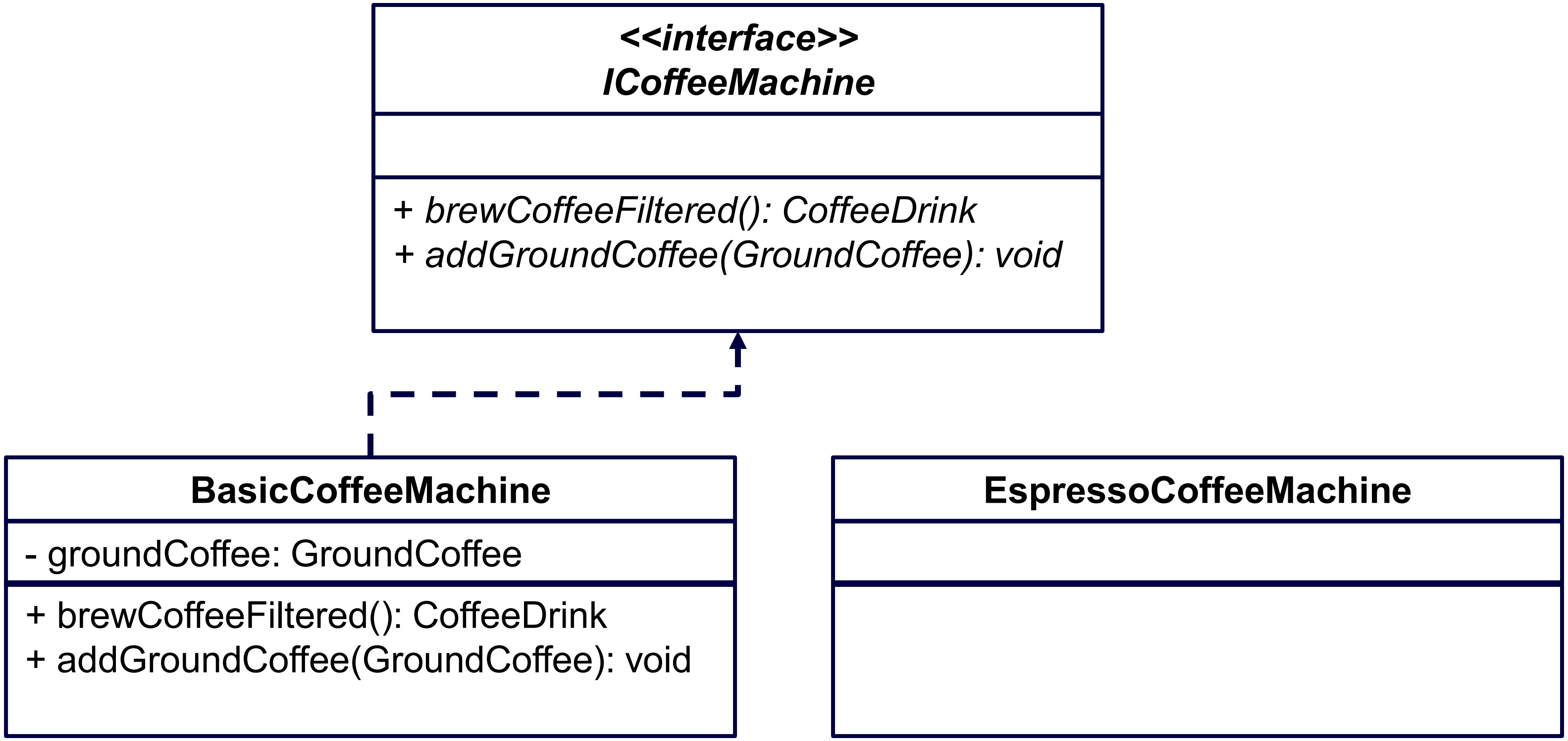
+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*

BasicCoffeeMachine

- groundCoffee: GroundCoffee

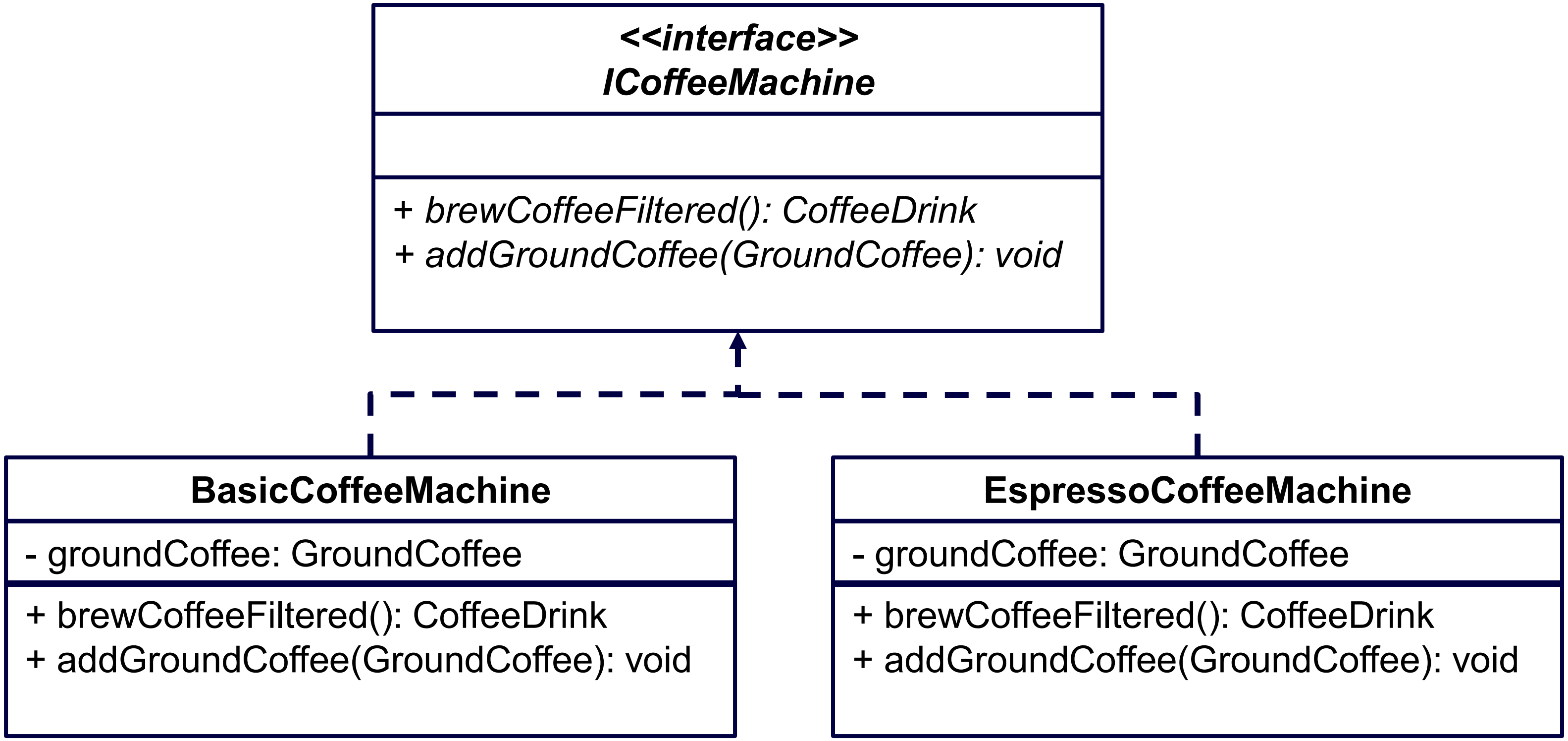
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void

Interface Segregation Principle

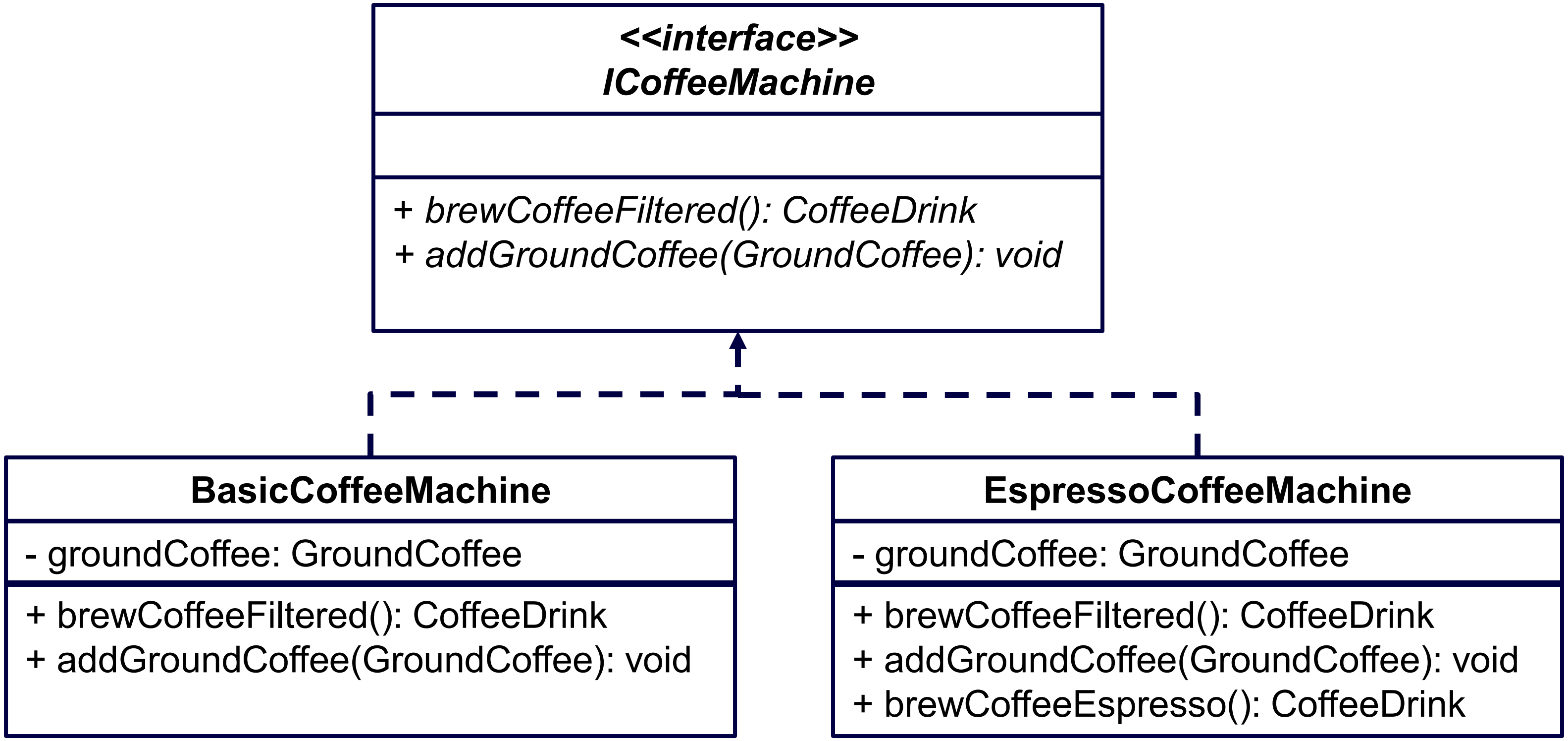


<<interface>>
ICoffeeMachine
+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
EspressoCoffeeMachine

Interface Segregation Principle

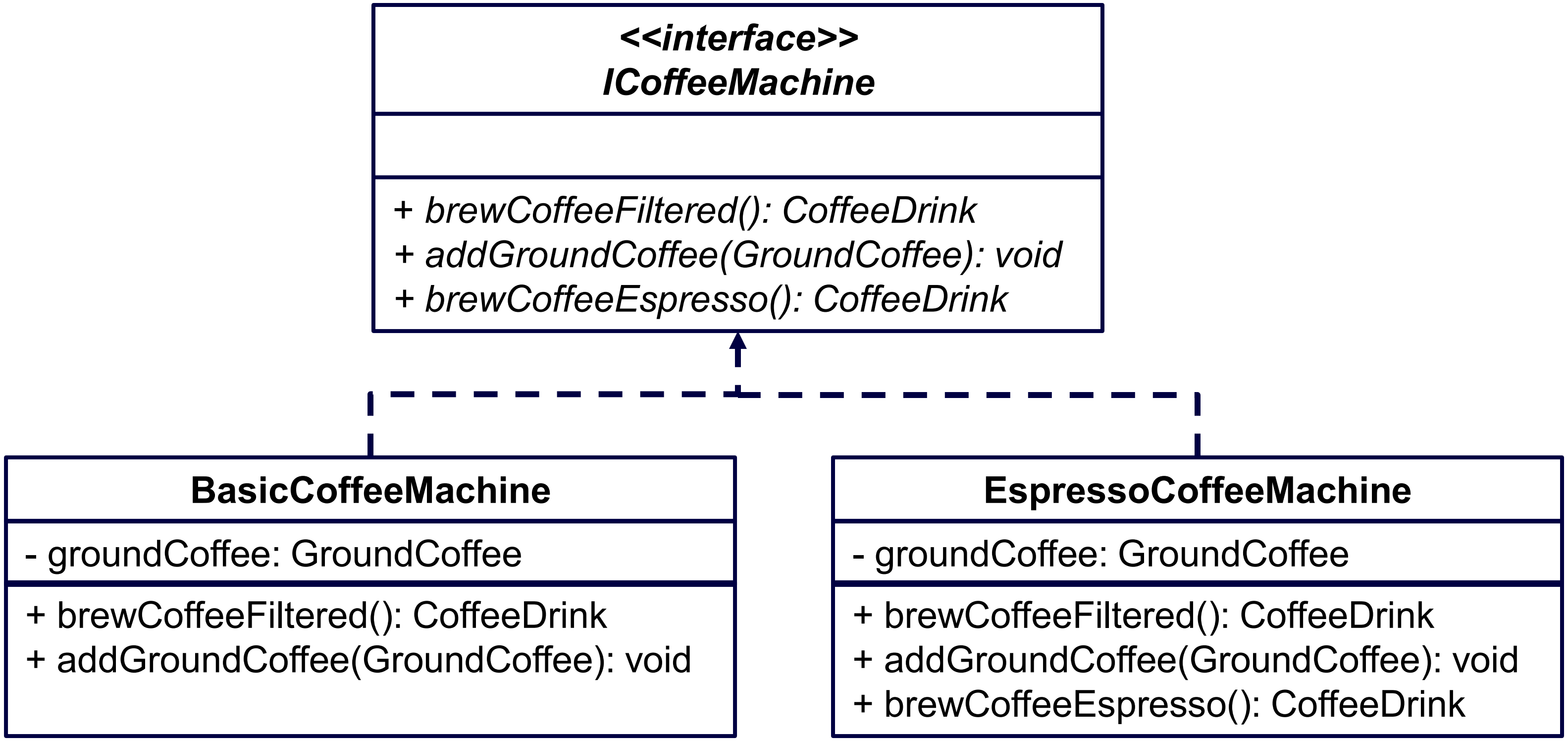

<<interface>>
ICoffeeMachine
+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void

Interface Segregation Principle



<<interface>>
ICoffeeMachine
+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*
+ *brewCoffeeEspresso(): CoffeeDrink*

BasicCoffeeMachine

- groundCoffee: GroundCoffee

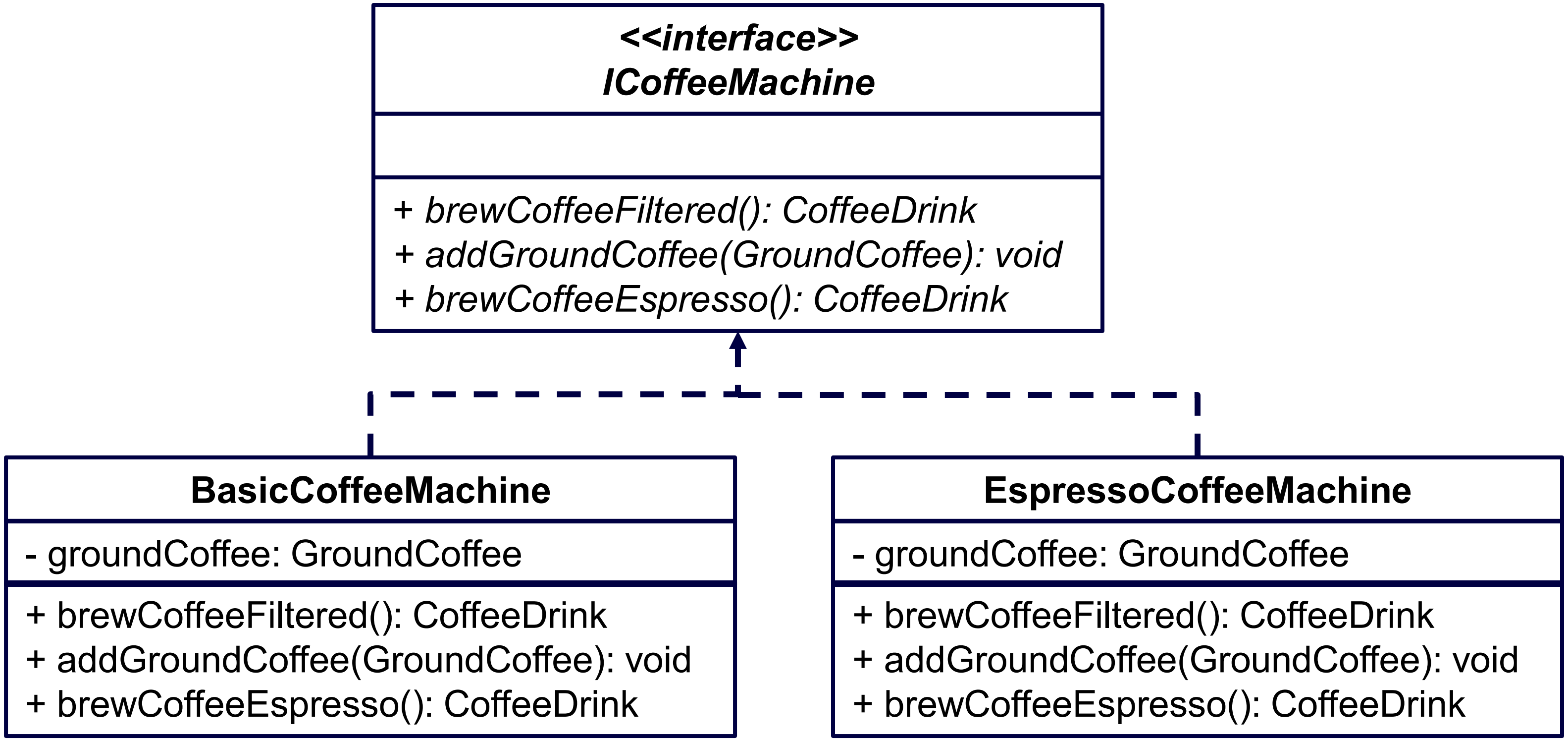
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void

EspressoCoffeeMachine

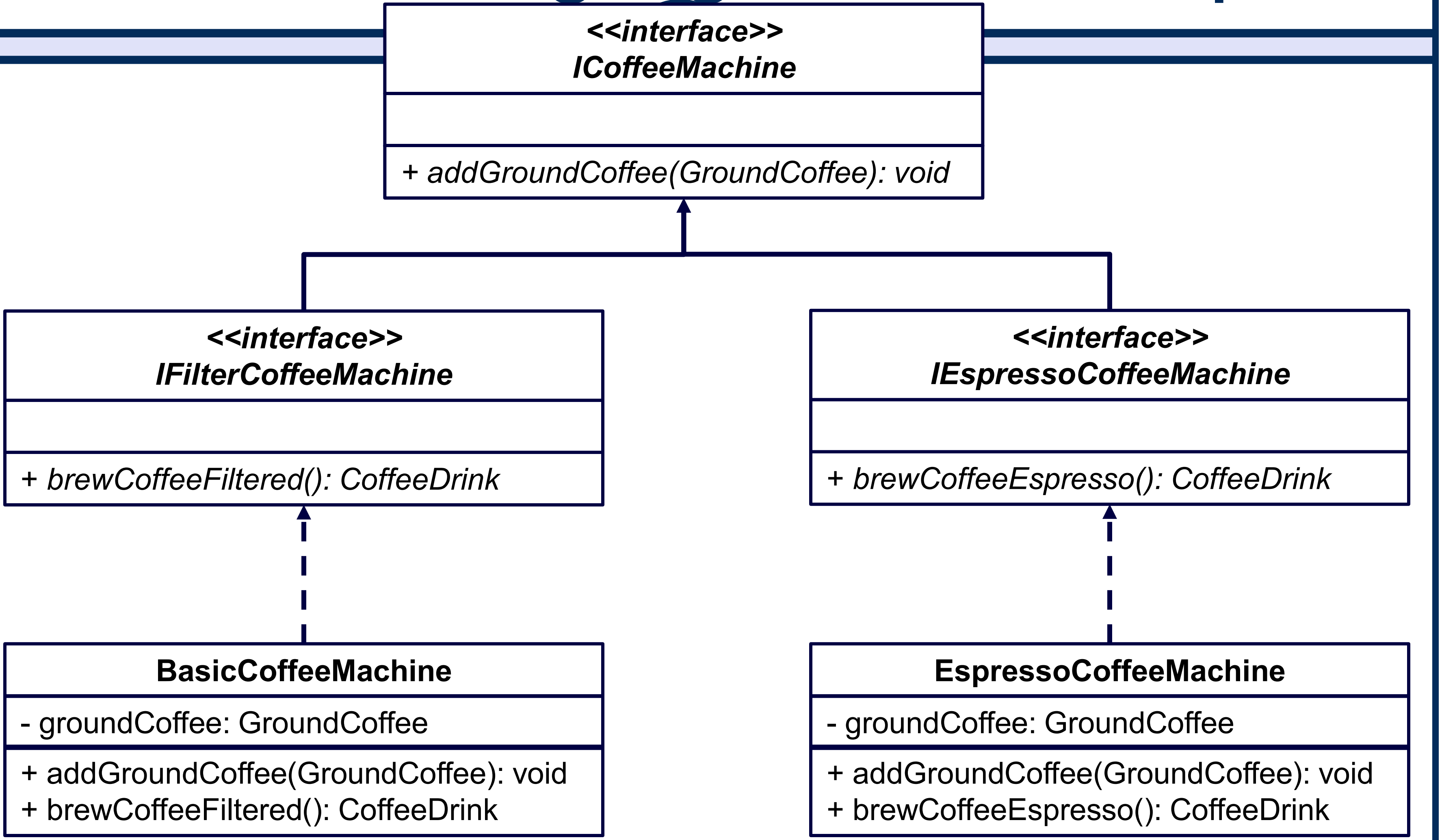
- groundCoffee: GroundCoffee

+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle


<<interface>>
ICoffeeMachine
+ *brewCoffeeFiltered(): CoffeeDrink*
+ *addGroundCoffee(GroundCoffee): void*
+ *brewCoffeeEspresso(): CoffeeDrink*
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ brewCoffeeFiltered(): CoffeeDrink
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ addGroundCoffee(GroundCoffee): void

<<interface>>
IFilterCoffeeMachine

+ brewCoffeeFiltered(): CoffeeDrink

<<interface>>
IEspressoCoffeeMachine

+ brewCoffeeEspresso(): CoffeeDrink

BasicCoffeeMachine

- groundCoffee: GroundCoffee

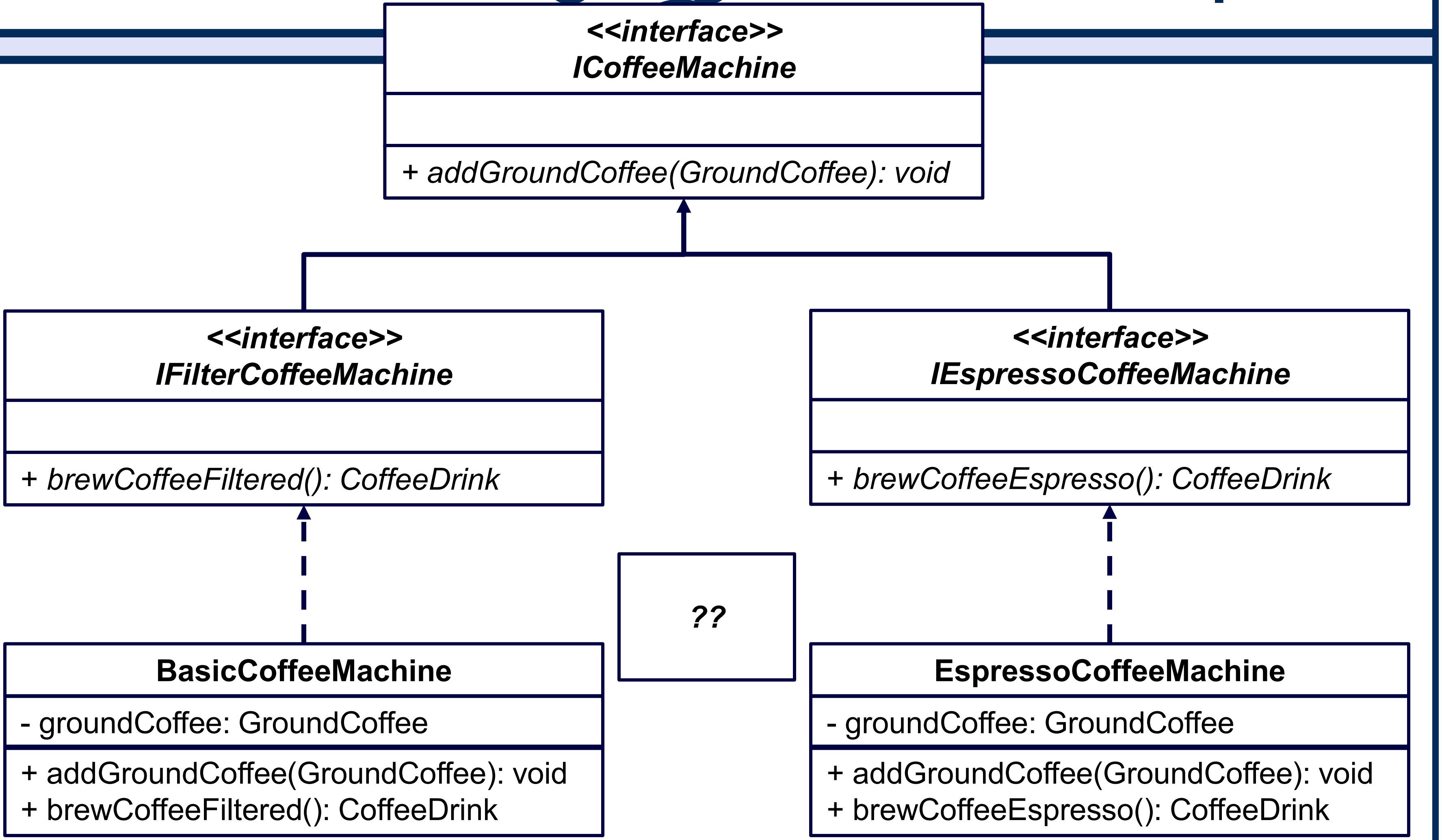
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink

EspressoCoffeeMachine

- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ addGroundCoffee(GroundCoffee): void

<<interface>>
IFilterCoffeeMachine

+ brewCoffeeFiltered(): CoffeeDrink

<<interface>>
IEspressoCoffeeMachine

+ brewCoffeeEspresso(): CoffeeDrink

BasicCoffeeMachine

- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink

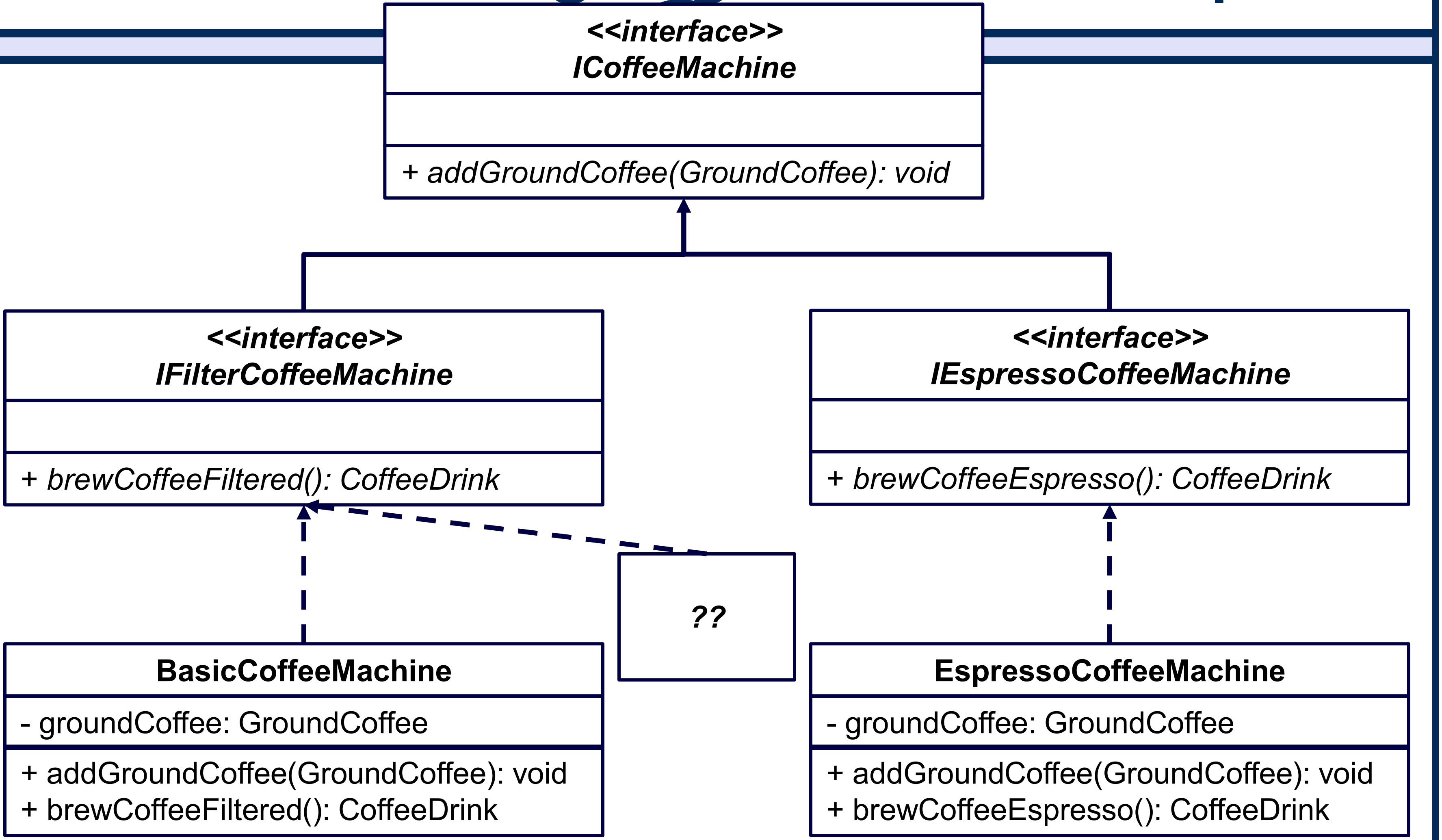
??

EspressoCoffeeMachine

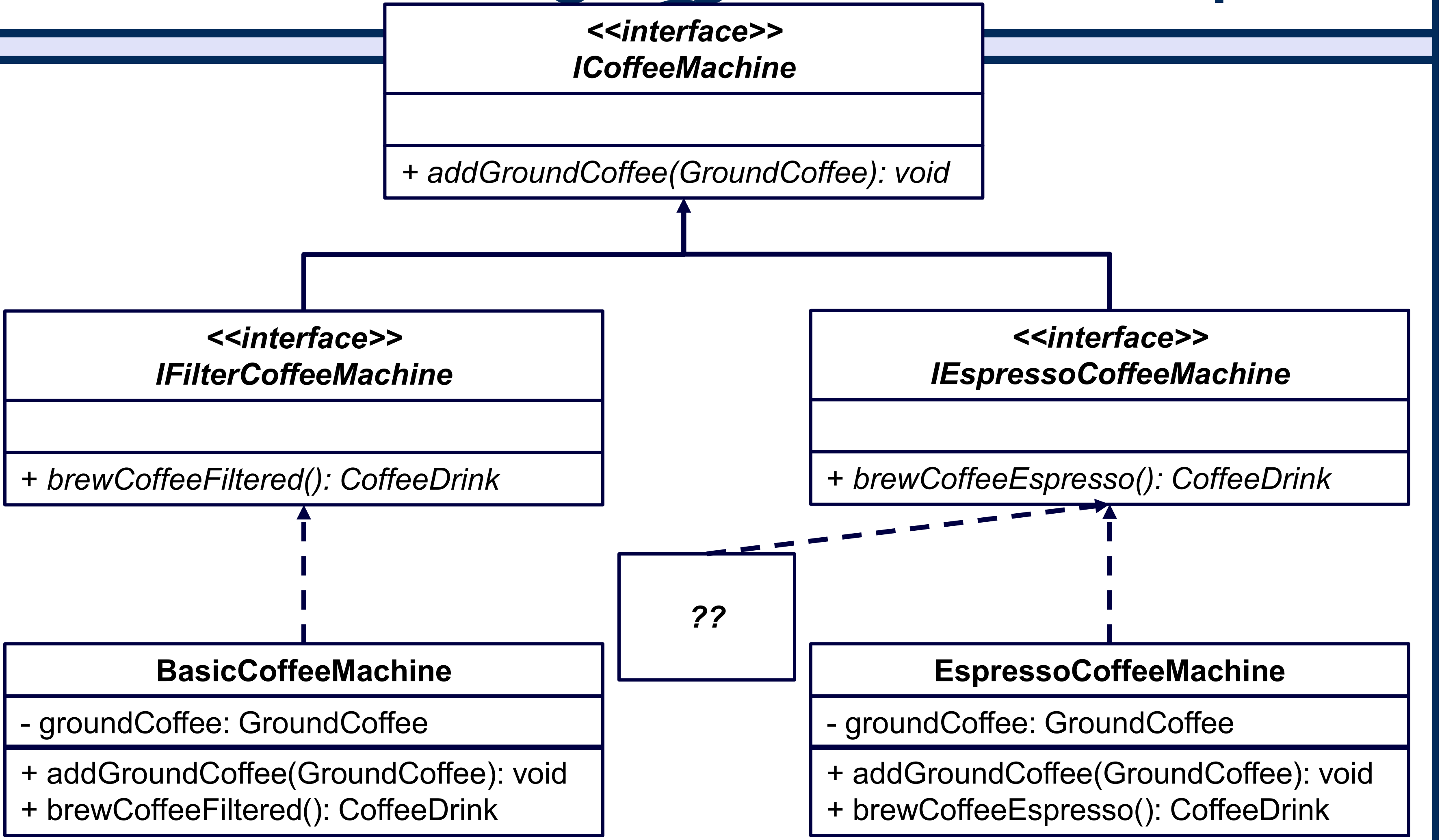
- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle


<<interface>>
ICoffeeMachine
+ addGroundCoffee(GroundCoffee): void
<<interface>>
IFilterCoffeeMachine
+ brewCoffeeFiltered(): CoffeeDrink
<<interface>>
IEspressoCoffeeMachine
+ brewCoffeeEspresso(): CoffeeDrink
??
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ addGroundCoffee(GroundCoffee): void

<<interface>>
IFilterCoffeeMachine

+ brewCoffeeFiltered(): CoffeeDrink

<<interface>>
IEspressoCoffeeMachine

+ brewCoffeeEspresso(): CoffeeDrink

??

BasicCoffeeMachine

- groundCoffee: GroundCoffee

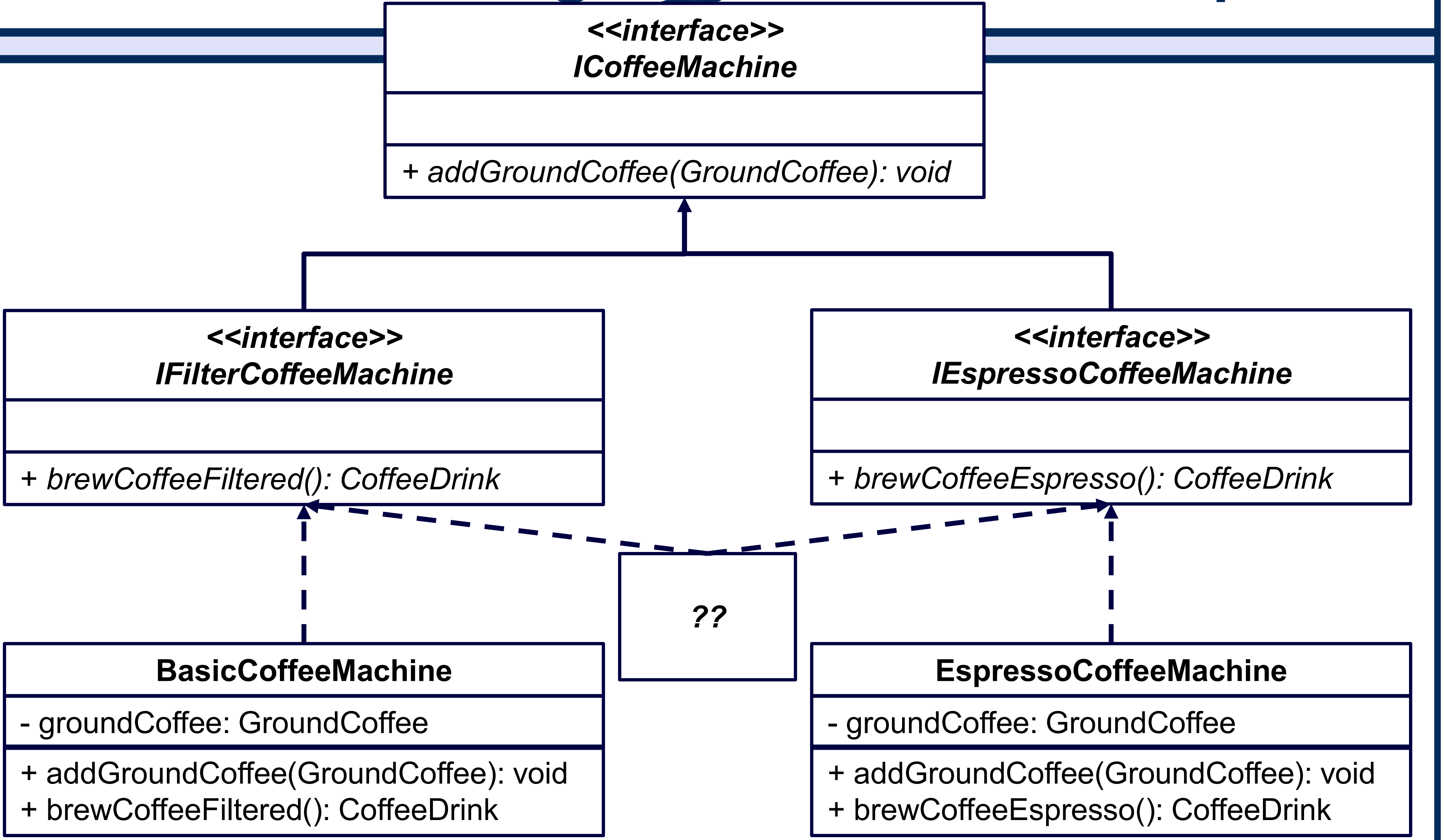
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink

EspressoCoffeeMachine

- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ addGroundCoffee(GroundCoffee): void

<<interface>>
IFilterCoffeeMachine

+ brewCoffeeFiltered(): CoffeeDrink

<<interface>>
IEspressoCoffeeMachine

+ brewCoffeeEspresso(): CoffeeDrink

??

BasicCoffeeMachine

- groundCoffee: GroundCoffee

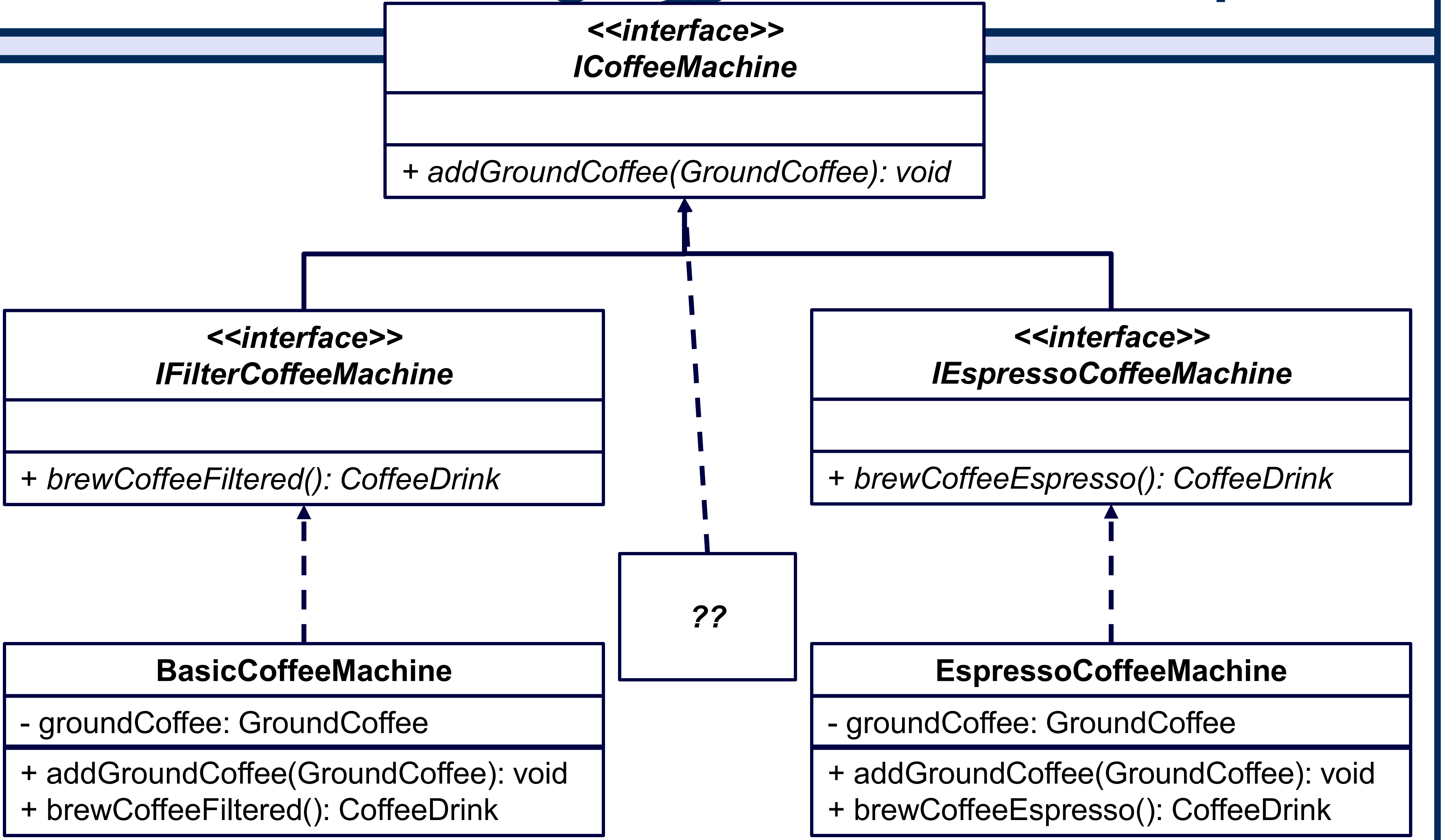
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink

EspressoCoffeeMachine

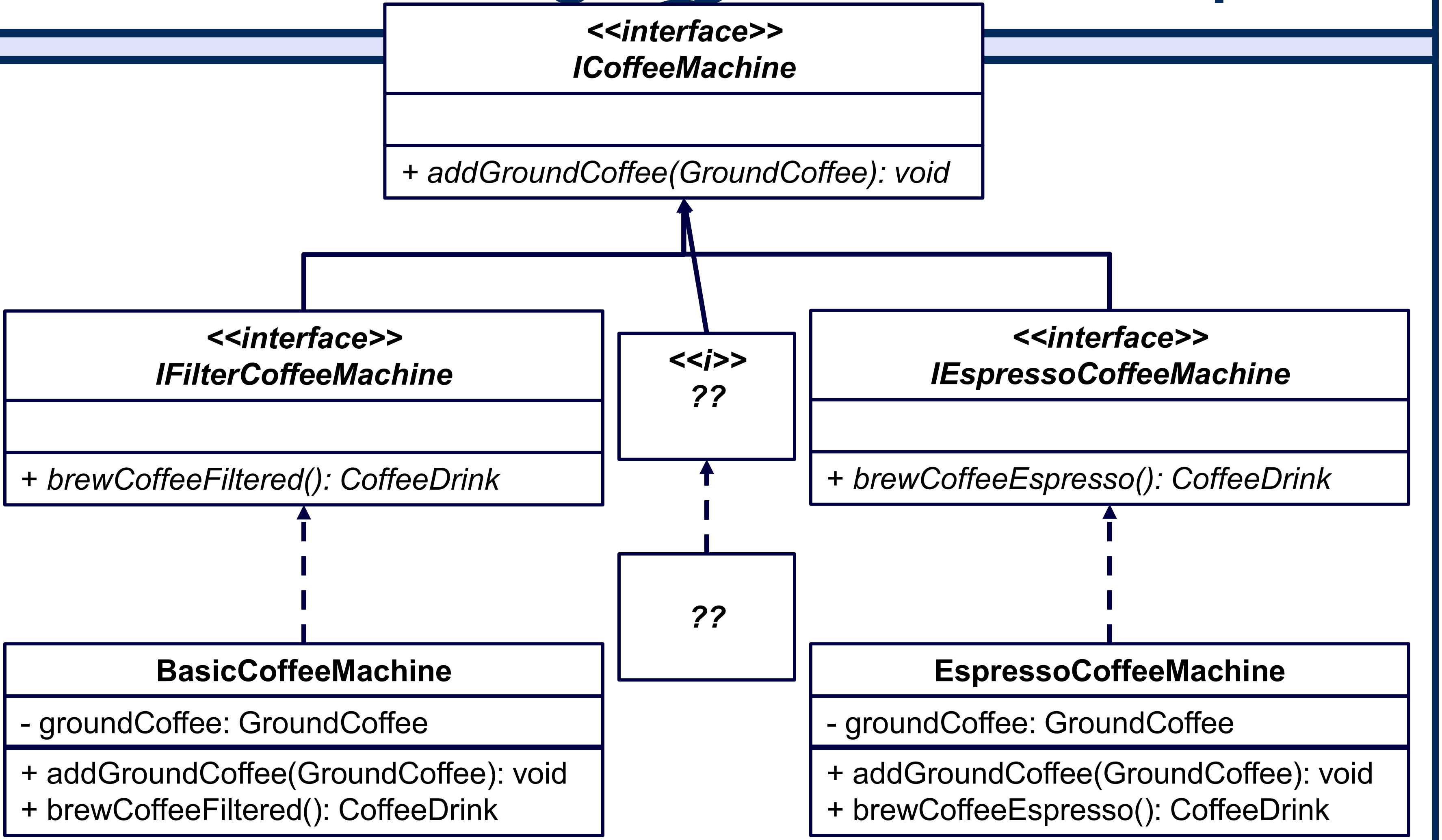
- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle


<<interface>>
ICoffeeMachine
+ addGroundCoffee(GroundCoffee): void
<<interface>>
IFilterCoffeeMachine
+ brewCoffeeFiltered(): CoffeeDrink
<<interface>>
IEspressoCoffeeMachine
+ brewCoffeeEspresso(): CoffeeDrink
??
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine

+ addGroundCoffee(GroundCoffee): void

<<interface>>
IFilterCoffeeMachine

+ brewCoffeeFiltered(): CoffeeDrink

<<i>>
??

??

<<interface>>
IEspressoCoffeeMachine

+ brewCoffeeEspresso(): CoffeeDrink

BasicCoffeeMachine

- groundCoffee: GroundCoffee

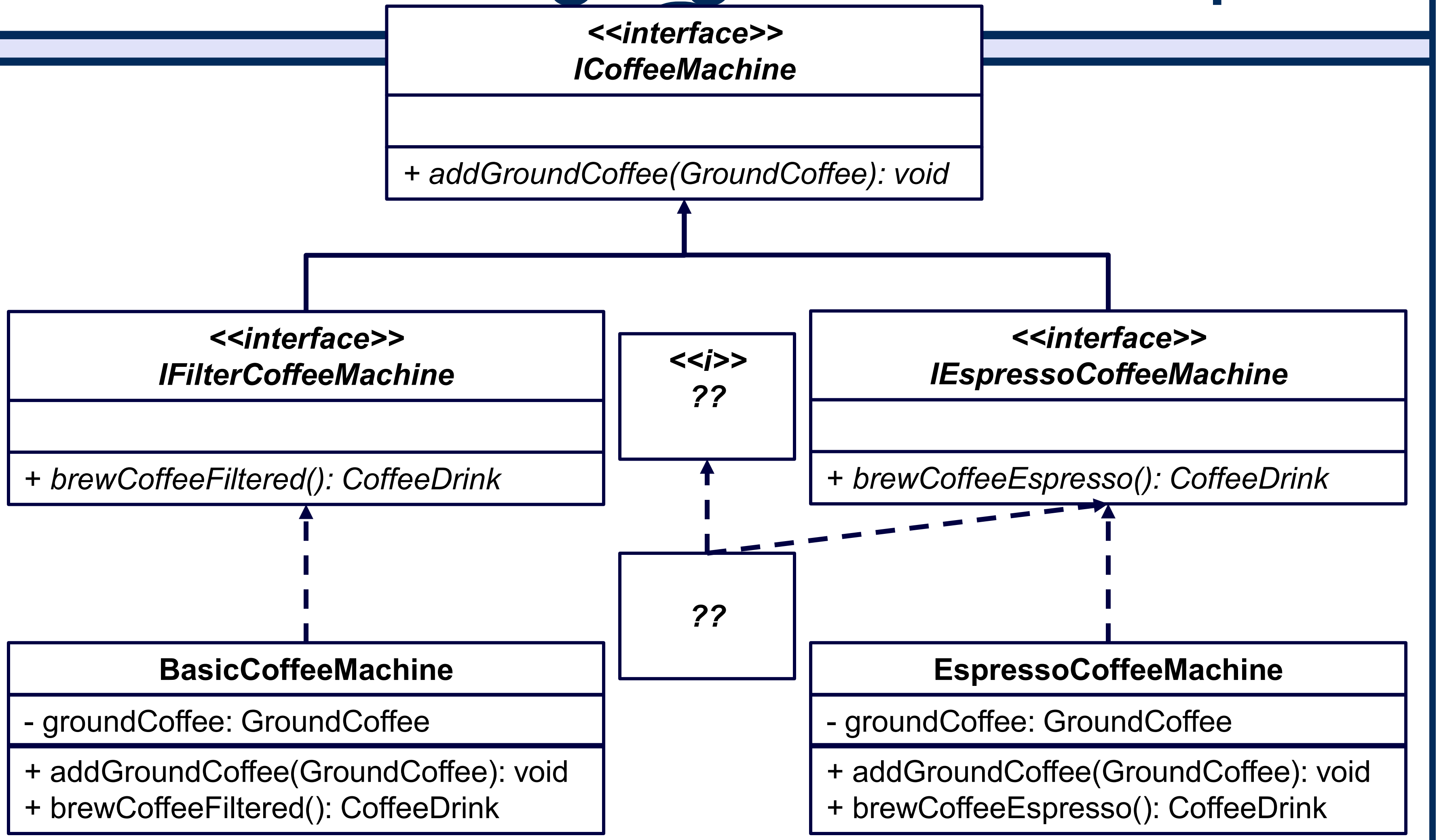
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink

EspressoCoffeeMachine

- groundCoffee: GroundCoffee

+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

Interface Segregation Principle



<<interface>>
ICoffeeMachine
+ addGroundCoffee(GroundCoffee): void
<<interface>>
IFilterCoffeeMachine
+ brewCoffeeFiltered(): CoffeeDrink
<<i>>
??
<<interface>>
IEspressoCoffeeMachine
+ brewCoffeeEspresso(): CoffeeDrink
BasicCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeFiltered(): CoffeeDrink
??
EspressoCoffeeMachine
- groundCoffee: GroundCoffee
+ addGroundCoffee(GroundCoffee): void
+ brewCoffeeEspresso(): CoffeeDrink

On Tap For Today

- Abstract Classes
- SOLID Principles
- Practice

To Do For Next Time

- Be working on Set5
- Be working on Final Project