Question 1: Refactor a Multi-Function Printer Interface

Refactor the `IMultiFunctionPrinter` interface into smaller, more specific interfaces to adhere to ISP. Ensure that classes only implement the methods they need.

```
from abc import ABC, abstractmethod

# Segregated Interfaces
class IPrinter(ABC):
    @abstractmethod
    def print(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan(self, document):
        pass

class IFax(ABC):
    @abstractmethod
    def fax(self, document):
        pass

# Implementing only required functionality
class BasicPrinter(IPrinter):
    def print(self, document):
        print(f"Printing: {document}")

class MultiFunctionPrinter(IPrinter, IScanner, IFax):
    def print(self, document):
        print(f"Printing: {document}")

    def scan(self, document):
        print(f"Scanning: {document}")

    def fax(self, document):
        print(f"Faxing: {document}")

# Testing
printer = BasicPrinter()
printer.print("Test Document")

mfp = MultiFunctionPrinter()
mfp.scan("Important Document")
mfp.fax("Confidential Fax")
```

```
Printing: Test Document
Scanning: Important Document
Faxing: Confidential Fax
```

Question 2: Refactor a Vehicle Interface

Refactor the `IVehicle` interface into smaller, more specific interfaces to adhere to ISP. Ensure that classes only implement the methods they need.

```
from abc import ABC, abstractmethod

# Segregated Interfaces
class IDrivable(ABC):
    @abstractmethod
    def drive(self):
        pass

class IFlyable(ABC):
    @abstractmethod
    def fly(self):
        pass

class ISailable(ABC):
    @abstractmethod
    def sail(self):
        pass

# Implementing only required functionality
class Car(IDrivable):
```

```
    def drive(self):
        print("Driving on the road")

class Airplane(IFlyable):
    def fly(self):
        print("Flying in the sky")

class Boat(ISailable):
    def sail(self):
        print("Sailing on the water")

# Testing
car = Car()
car.drive()

plane = Airplane()
plane.fly()

boat = Boat()
boat.sail()
```

```
Driving on the road
Flying in the sky
Sailing on the water
```

Question 3: Refactor a Payment Gateway Interface

Refactor the `IPaymentGateway` interface into smaller, more specific interfaces to adhere to ISP. Ensure that classes only implement the methods they need.

```
from abc import ABC, abstractmethod

# Segregated Interfaces
class ICreditCardPayment(ABC):
    @abstractmethod
    def process_credit_card(self, amount):
        pass

class IPayPalPayment(ABC):
    @abstractmethod
    def process_paypal(self, amount):
        pass

class ICryptoPayment(ABC):
    @abstractmethod
    def process_crypto(self, amount):
        pass

# Implementing only required functionality
class CreditCardGateway(ICreditCardPayment):
    def process_credit_card(self, amount):
        print(f"Processing credit card payment: ${amount}")

class PayPalGateway(IPayPalPayment):
    def process_paypal(self, amount):
        print(f"Processing PayPal payment: ${amount}")

class CryptoGateway(ICryptoPayment):
    def process_crypto(self, amount):
        print(f"Processing cryptocurrency payment: ${amount}")

# Testing
cc_processor = CreditCardGateway()
cc_processor.process_credit_card(100)

paypal_processor = PayPalGateway()
paypal_processor.process_paypal(200)

crypto_processor = CryptoGateway()
crypto_processor.process_crypto(300)
```

```
Processing credit card payment: $100
Processing PayPal payment: $200
Processing cryptocurrency payment: $300
```