```
import asyncio
from dataclasses import dataclass
from enum import Enum
from typing import List, Optional

from swarms import Agent


class InsuranceType(Enum):
    AUTO = "auto"
    LIFE = "life"
    HEALTH = "health"
    HOME = "home"
    BUSINESS = "business"
    DENTAL = "dental"
    TRAVEL = "travel"


@dataclass
class InsuranceProduct:
    code: str
    name: str
    type: InsuranceType
    description: str
    coverage: List[str]
    price_range: str
```

```
    min_coverage: float
    max_coverage: float
    payment_options: List[str]
    waiting_period: str
    available: bool


# Simulated product database
INSURANCE_PRODUCTS = {
    "AUTO001": InsuranceProduct(
        code="AUTO001",
        name="Seguro Auto Total",
        type=InsuranceType.AUTO,
        description="Seguro completo para vehículos con cobertura integral",
        coverage=[
            "Daños por colisión",
            "Robo total",
            "Responsabilidad civil",
            "Asistencia en carretera 24/7",
            "Gastos médicos ocupantes",
        ],
        price_range="$800-2000 USD/año",
        min_coverage=10000,
        max_coverage=50000,
        payment_options=["Mensual", "Trimestral", "Anual"],
        waiting_period="Inmediata",
```

```
    available=True,
),
"LIFE001": InsuranceProduct(
    code="LIFE001",
    name="Vida Protegida Plus",
    type=InsuranceType.LIFE,
    description="Seguro de vida con cobertura extendida y beneficios adicionales",
    coverage=[
        "Muerte natural",
        "Muerte accidental (doble indemnización)",
        "Invalidez total y permanente",
        "Enfermedades graves",
        "Gastos funerarios",
    ],
    price_range="$30-100 USD/mes",
    min_coverage=50000,
    max_coverage=1000000,
    payment_options=["Mensual", "Anual"],
    waiting_period="30 días",
    available=True,
),
"HEALTH001": InsuranceProduct(
    code="HEALTH001",
    name="Salud Preferencial",
    type=InsuranceType.HEALTH,
    description="Plan de salud premium con cobertura internacional",
```

```
        coverage=[
            "Hospitalización",
            "Cirugías",
            "Consultas médicas",
            "Medicamentos",
            "Tratamientos especializados",
            "Cobertura internacional",
        ],
        price_range="$100-300 USD/mes",
        min_coverage=100000,
        max_coverage=5000000,
        payment_options=["Mensual", "Anual"],
        waiting_period="90 días",
        available=True,
    ),
}


class WorkflowNode(Enum):
    MAIN_MENU = "main_menu"
    CHECK_AVAILABILITY = "check_availability"
    PRODUCT_DETAILS = "product_details"
    QUOTE_REQUEST = "quote_request"
    CLAIMS = "claims"
    LOCATE_OFFICE = "locate_office"
    PAYMENT_OPTIONS = "payment_options"
```

```
LATAM_LOCATIONS = {
    "Brasil": [
        {
            "city": "São Paulo",
            "offices": [
                {
                    "address": "Av. Paulista, 1374 - Bela Vista",
                    "phone": "+55 11 1234-5678",
                    "hours": "Lun-Vie: 9:00-18:00",
                }
            ],
        }
    ],
    "México": [
        {
            "city": "Ciudad de México",
            "offices": [
                {
                    "address": "Paseo de la Reforma 250, Juárez",
                    "phone": "+52 55 1234-5678",
                    "hours": "Lun-Vie: 9:00-18:00",
                }
            ],
        }
```

```
    ],
}



class InsuranceBot:
    def __init__(self):
        self.agent = Agent(
            agent_name="LATAM-Insurance-Agent",
                system_prompt="""You are a specialized insurance assistant for Latin America's leading insurance provider.

Key Responsibilities:
1. Product Information:
   - Explain our comprehensive insurance portfolio
   - Provide detailed coverage information
   - Compare plans and benefits
   - Quote estimates based on customer needs

2. Customer Service:
   - Process policy inquiries
   - Handle claims information
   - Assist with payment options
   - Locate nearest offices

3. Cultural Considerations:
   - Communicate in Spanish and Portuguese
```

- Understand LATAM insurance regulations
- Consider regional healthcare systems
- Respect local customs and practices

Use the following simulated product database for accurate information:
{INSURANCE_PRODUCTS}

When discussing products, always reference accurate prices, coverage amounts, and waiting periods.""",
            model_name="gpt-4",
            max_loops=1,
            verbose=True,
)

        self.current_node = WorkflowNode.MAIN_MENU
        self.current_product = None

    async def process_user_input(self, user_input: str) -> str:
        """Process user input and return appropriate response"""
        try:
            if self.current_node == WorkflowNode.MAIN_MENU:
                menu_choice = user_input.strip()

                if menu_choice == "1":
                    # Use agent to provide personalized product recommendations
                    return await self.agent.run(

```
                """Por favor ayude al cliente a elegir un producto:

Productos disponibles:
- AUTO001: Seguro Auto Total
- LIFE001: Vida Protegida Plus
- HEALTH001: Salud Preferencial

Explique brevemente cada uno y solicite información sobre sus necesidades específicas."""
                )

            elif menu_choice == "2":
                self.current_node = WorkflowNode.QUOTE_REQUEST
                # Use agent to handle quote requests
                return await self.agent.run(
                    """Inicie el proceso de cotización.
                Solicite la siguiente información de manera conversacional:
                1. Tipo de seguro
                2. Información personal básica
                3. Necesidades específicas de cobertura"""
                )

            elif menu_choice == "3":
                return await self.agent.run(
                    """Explique el proceso de reclamos para cada tipo de seguro,
                incluyendo documentación necesaria y tiempos estimados."""
                )
```

```
elif menu_choice == "4":
    self.current_node = WorkflowNode.LOCATE_OFFICE
    # Use agent to provide location guidance
    return await self.agent.run(
        f"""Based on our office locations: {LATAM_LOCATIONS}
    Ask the customer for their location and help them find the nearest office.
    Provide the response in Spanish."""
    )

elif menu_choice == "5":
    # Use agent to explain payment options
    return await self.agent.run(
        """Explique todas las opciones de pago disponibles,
    incluyendo métodos, frecuencias y cualquier descuento por pago anticipado."""
    )

elif menu_choice == "6":
    # Use agent to handle advisor connection
    return await self.agent.run(
        """Explique el proceso para conectar con un asesor personal,
    horarios de atención y canales disponibles."""
    )

else:
    return await self.agent.run(
```

```
            "Explain that the option is invalid and list the main menu options."
        )

elif self.current_node == WorkflowNode.LOCATE_OFFICE:
    # Use agent to process location request
    return await self.agent.run(
        f"""Based on user input: '{user_input}'
    and our office locations: {LATAM_LOCATIONS}
    Help them find the most relevant office. Response in Spanish."""
    )

# Check if input is a product code
if user_input.upper() in INSURANCE_PRODUCTS:
    product = self.get_product_info(user_input.upper())
    # Use agent to provide detailed product information
    return await self.agent.run(
        f"""Provide detailed information about this product:
    {self.format_product_info(product)}
    Include additional benefits and comparison with similar products.
    Response in Spanish."""
    )

# Handle general queries
return await self.agent.run(
    f"""The user said: '{user_input}'
Provide a helpful response based on our insurance products and services.
```

```
        Response in Spanish."""
        )

    except Exception:
        self.current_node = WorkflowNode.MAIN_MENU
        return await self.agent.run(
            "Explain that there was an error and list the main menu options. Response in Spanish."
        )

def get_product_info(
    self, product_code: str
) -> Optional[InsuranceProduct]:
    """Get product information from simulated database"""
    return INSURANCE_PRODUCTS.get(product_code)

def format_product_info(self, product: InsuranceProduct) -> str:
    """Format product information for display"""
    return f"""
    Producto: {product.name} (Código: {product.code})
    Tipo: {product.type.value}
    Descripción: {product.description}

    Cobertura incluye:
    {chr(10).join(f'- {coverage}' for coverage in product.coverage)}

    Rango de precio: {product.price_range}
```

```
        Cobertura mínima: ${product.min_coverage:,.2f} USD
        Cobertura máxima: ${product.max_coverage:,.2f} USD

        Opciones de pago: {', '.join(product.payment_options)}
        Período de espera: {product.waiting_period}
        Estado: {'Disponible' if product.available else 'No disponible'}
        """

    def handle_main_menu(self) -> List[str]:
        """Return main menu options"""
        return [
            "1. Consultar productos de seguro",
            "2. Solicitar cotización",
            "3. Información sobre reclamos",
            "4. Ubicar oficina más cercana",
            "5. Opciones de pago",
            "6. Hablar con un asesor",
        ]


async def main():
    """Run the interactive session"""
    bot = InsuranceBot()

    print(
        "Sistema de Seguros LATAM inicializado. Escriba 'salir' para terminar."
```

```
    )
    print("\nOpciones disponibles:")
    print("\n".join(bot.handle_main_menu()))

    while True:
        user_input = input("\nUsted: ").strip()

        if user_input.lower() in ["salir", "exit"]:
            print("¡Gracias por usar nuestro servicio!")
            break

        response = await bot.process_user_input(user_input)
        print(f"Agente: {response}")


if __name__ == "__main__":
    asyncio.run(main())
```