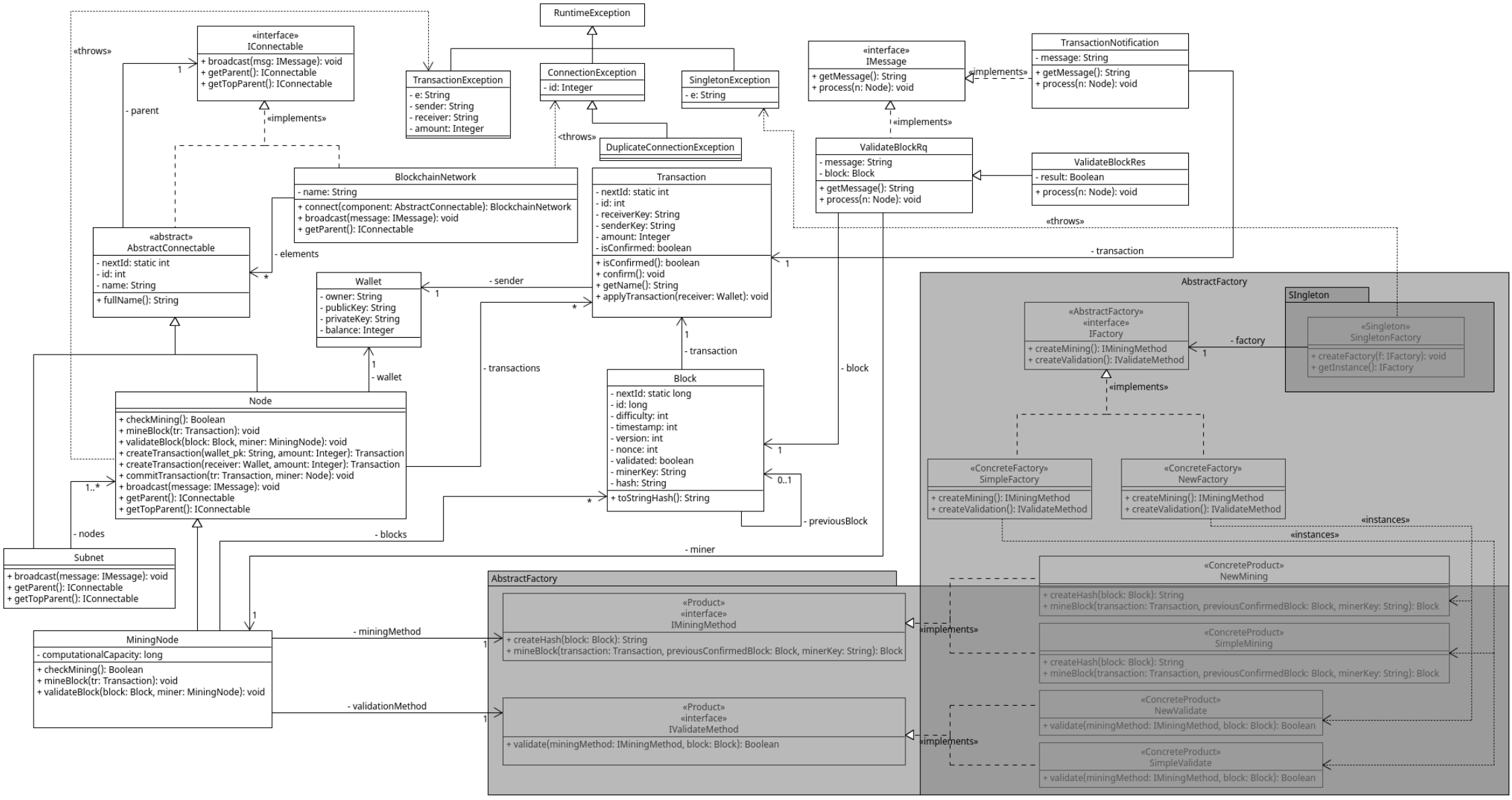
RuntimeException
«interface»
IConnectable
+ broadcast(msg: IMessage): void
+ getParent(): IConnectable
+ getTopParent(): IConnectable
«throws»
1
- parent
«implements»
TransactionException
- e: String
- sender: String
- receiver: String
- amount: Integer
ConnectionException
- id: Integer
SingletonException
- e: String
«interface»
IMessage
+ getMessage(): String
+ process(n: Node): void
«implements»
TransactionNotification
- message: String
+ getMessage(): String
+ process(n: Node): void
<throws>
DuplicateConnectionException
«implements»
ValidateBlockRq
- message: String
- block: Block
+ getMessage(): String
+ process(n: Node): void
ValidateBlockRes
- result: Boolean
+ process(n: Node): void
«throws»
BlockchainNetwork
- name: String
+ connect(component: AbstractConnectable): BlockchainNetwork
+ broadcast(message: IMessage): void
+ getParent(): IConnectable
Transaction
- nextId: static int
- id: int
- receiverKey: String
- senderKey: String
- amount: Integer
- isConfirmed: boolean
+ isConfirmed(): boolean
+ confirm(): void
+ getName(): String
+ applyTransaction(receiver: Wallet): void
- transaction
«abstract»
AbstractConnectable
- nextId: static int
- id: int
- name: String
+ fullName(): String
- elements
*
Wallet
- owner: String
- publicKey: String
- privateKey: String
- balance: Integer
- sender
1
*
1
- wallet
- transactions
- block
Node
+ checkMining(): Boolean
+ mineBlock(tr: Transaction): void
+ validateBlock(block: Block, miner: MiningNode): void
+ createTransaction(wallet_pk: String, amount: Integer): Transaction
+ createTransaction(receiver: Wallet, amount: Integer): Transaction
+ commitTransaction(tr: Transaction, miner: Node): void
+ broadcast(message: IMessage): void
+ getParent(): IConnectable
+ getTopParent(): IConnectable
1..*
Block
- nextId: static long
- id: long
- difficulty: int
- timestamp: int
- version: int
- nonce: int
- validated: boolean
- minerKey: String
- hash: String
+ toStringHash(): String
*
1
0..1
- previousBlock
- nodes
- blocks
- miner
Subnet
+ broadcast(message: IMessage): void
+ getParent(): IConnectable
+ getTopParent(): IConnectable
1
MiningNode
- computationalCapacity: long
+ checkMining(): Boolean
+ mineBlock(tr: Transaction): void
+ validateBlock(block: Block, miner: MiningNode): void
- miningMethod
- validationMethod
1
1
AbstractFactory
«Product»
«interface»
IMiningMethod
+ createHash(block: Block): String
+ mineBlock(transaction: Transaction, previousConfirmedBlock: Block, minerKey: String): Block
«Product»
«interface»
IValidateMethod
+ validate(miningMethod: IMiningMethod, block: Block): Boolean
AbstractFactory
SIngleton
«AbstractFactory»
«interface»
IFactory
+ createMining(): IMiningMethod
+ createValidation(): IValidateMethod
- factory
1
«Singleton»
SingletonFactory
+ createFactory(f: IFactory): void
+ getInstance(): IFactory
«implements»
«ConcreteFactory»
SimpleFactory
+ createMining(): IMiningMethod
+ createValidation(): IValidateMethod
«ConcreteFactory»
NewFactory
+ createMining(): IMiningMethod
+ createValidation(): IValidateMethod
«instances»
«instances»
«implements»
«implements»
«ConcreteProduct»
NewMining
+ createHash(block: Block): String
+ mineBlock(transaction: Transaction, previousConfirmedBlock: Block, minerKey: String): Block
«ConcreteProduct»
SimpleMining
+ createHash(block: Block): String
+ mineBlock(transaction: Transaction, previousConfirmedBlock: Block, minerKey: String): Block
«ConcreteProduct»
NewValidate
+ validate(miningMethod: IMiningMethod, block: Block): Boolean
«ConcreteProduct»
SimpleValidate
+ validate(miningMethod: IMiningMethod, block: Block): Boolean

Para este diseño empezamos definiendo la clase AbstractConnectable, esta clase nos permite montar la jerarquía de nodos y subredes, dicha clase implementa la interfaz IConnectable al igual que la clase BlockchainNetwork, de esta forma podremos implementar la comunicación entre los elementos del sistema, mediante invocaciones al método 'broadcast', este método tiene versiones distintas dependiendo del tipo de elemento, BlockchainNetwork reenvía el mensaje a todos los elementos de su lista de elementos, las subredes lo procesan y reenvían a su lista de nodos mientras qué los nodos solo procesan. Como cada mensaje implementa la interfaz IMessage, el método process de esta nos permite implementar distintas funcionalidades en función del tipo de mensaje y de quien lo procesa, p.e. si un nodo procesa un mensaje validateBlockRes, commiteará la transacción en su lista de transacciones. MiningNode tiene la misma funcionalidad que Node pero con alguna función extra, sus métodos validateBlock y mineBlock invocan a los métodos respectivos de las clases concretas que implementan las interfaces de minado y validación.

Todas las excepciones del diseño extienden RuntimeException para no tener que hacer catch cada vez que se invoca un método que la lance, DuplicateConnectionException hereda de ConnectionException porque recibe los mismos parámetros que su clase padre para ser construida.

Para el apartado de extraordinaria, se han usado los patrones AbstractFactory y Singleton, abstract factory nos permite esa delegación de responsabilidades e instanciar diversos objetos sin necesidad de saber cual, en este caso, instanciar una fábrica del método simple de minado, provoca que el método de validación sea instanciado por esa misma fábrica, sin posibilidad de romper la pareja. Sin embargo, este diseño no evitaba que todos los componentes de la red usen la misma fábrica, por tanto, usamos el Singleton para hacer que haya una única instancia de la fábrica, cubriendo ese problema (se ha implementado una excepción para este patrón para evitar errores si no se usa correctamente). Hay que tener en cuenta que este diseño no evita instanciar un método de minado/validación sin usar la fábrica, implementar eso provocaría el mal funcionamiento del apartado 3, en conclusión, si las instancias se hacen con abstract factory y singleton, nunca podrá haber distintos pares de minado-validación.