Here's the code I have already:
https://github.com/awang193/Java-Procedural-Generation-Test

GUI

public static void main(String[] args)
 	
public void start()
	// Main method
	
	// Starts the GUI

```
void start()
     Make new group
     Add map layer and then player/monster layer

     While player isn't dead
          Update map layer and player monster layer using a timer
```

Character

protected final CHARACTER_W, CHARACTER_H;
protected Point2D.Double
protected int lvl;
protected int hp;
protected int damage
 	
protected Armor armor;
protected Weapon wpn;
	// Width, Height
	
	// Character position
	// Character Level
	// Health
	// Damage
	
	// Armor
	// Weapon

public void pickUp()
 	
public void getLevel(int lv)
 	
public void setLevel(int lv)
 	
public void getExp(int xp)
 	
public void setExp(int xp)
 	
public void getHp(int hp)
 	
public void addHp(int hp)
	// Pick up item
	
	//Obtain level status
	
	// Change level if exp requirement is met
	// Obtain exp from monsters/treasures
	// Reset exp to 0 if level up
	
	// Obtain HP value
	
	// Change HP from potions/level

public boolean use(Item item)	up or Enemy attack/traps // Use items
public boolean equip(Item item)	// Show weapons/equipment
public void move()	// Up/Down/Left/Right
public void attack(ArrayList<Monster> monsters)	// Calls to Combat

void move()
    Check for W A S D for orientation
    Increase player's position in selected direction unless it could hit a wall

void attack(ArrayList<Monster> monsters)
    Weapon.use(ArrayList<Monster> monsters)

Weapon

private int damage private int reach	// Damage value of the weapon // Weapon range
public void use()	// Attack

void use()
    Check if any monsters are in the hitbox
    Set health of any hit monsters to be lowered by damage

Monster

protected final WIDTH, HEIGHT;
protected int lvl;
protected int hp;
protected int exp;
protected int moveSpeed;
	// Width, Height
	// Enemy Level
	// Health
	// EXP it gives when death
	// Move speed

public void getLevel(int lv)	//Obtain level status
public void setLevel(int lv)	// Change level
public void getExp()	// return exp

public void setExp(int xp)	// Change xp level
public int getHp()	// Obtain HP value
public void addHp(int hp)	// Change HP from potions/level
public void trackPlayer(Character ch)	// Track player and move
public void attack(ArrayList<Character>)	// Attack player
public void setMoveSpeed(int speed)	// Set the monster movement speed
public int getMoveSpeed()	// Obtain the monster's movement speed

```
void trackPlayer(Character ch)
     If player is within certain distance
          Check player's orientation with respect to monster
          Draw line from monster to player and slowly move along
line


public void attack(ArrayList<Character>)
     For every player in the list within attack range
     Subtract damage from each player's health
```

Room

protected int x, y, w, h
protected int roomLevel
protected Point center
protected boolean cleared
 	
protected ArrayList<Monster> monsters
	// x and y coordinates of corners
	// level of the room
	// center of the room
	// whether all monsters have been killed
	// List of room's monsters

public int getX()
public int getY()
public int getW()
public int getH()
public Point getCenter()
	//return x coordinate
	//return y coordinate
	//return the width
	//return the height
	//return the center

public boolean isCleared()	//return true or false if the room is cleared

public void setX()
public void setY()
public void setW()
public void setH()
public Point setCenter()
	//set the x coordinate
	//set the y coordinate
	//set the width
	//set the height
	//set center

```
public int getX()
      return x

public int getY()
      return y

public int getW()
      return w

public int getH()
      return h

public Point getCenter()
      return center

public boolean isCleared()
      return monsters.size() == 0

public void setX(int newX)
      x = newX

public void setY(int newY)
      y = newY

public void setW(int newW)
      w = newW

public void setH(int newH)
      h = newH

public void setCenter(Point newCenter)
      center = newCenter
```

BSPLeaf

private final int MIN_LEAF_SIZE
 	
private int, x, y, w, h
private Room room
private DungeonLeaf left, right
	// Minimum size for each dungeon "leaf"
	// X, Y, width, height of leaf
	// Room contained within leaf
	// Left and right sub-leaves
	// List of hallways leaf contains

public boolean split()
 	
public int getX()
public int getY()
public int getW()
public int getH()
public Room getRoom()
public BSPLeaf getLeft()
public BSPLeaf getRight()
public void setX(int x)
public void setY(int y)
public void setW(int w)
public void setH(int h)
public void setRoom(Room room)
public void setLeft(BSPLeaf left)
public void setRight(BSPLeaf right)
	// Method that splits a leaf into two sub-leaves, returns true if successful, false otherwise
	
	//return x coordinate
	//return y coordinate
	//return the width
	//return the height
	//return the room
	//return left BSPLeaf
	//return right BSPLeaf
	//set the x coordinate
	//set the y coordinate
	//set the width
	//set the height
	//set the room
	//set left BSPLeaf
	//set right BSPLeaf

```
public int getX()
     return x

public int getY()
     return y

public int getW()
     return w

public int getH()
     return h

public Room getRoom()
     return room

public BSPLeaf getLeft()
     return left

public BSPLeaf getRight()
```

```
      return right

public void setX(int newX)
      x = newX

public void setY(int newY)
      y = newY

public void setW(int newW)
      w = newW

public void setH(int newH)
      h = newH

public void setRoom(Room newRoom)
      room = newRoom

public void setLeft(BSPLeaf newLeft)
      left = newLeft

public void setRight(BSPLeaf right)
      right = newRight

public void split()
      if both left and right are null (hasn't split)
            randomly choose whether to split horiz or vert
            choose a spot to split at

            check if leafs are too small, if so resplit

            set left and right to new leafs

            recursively call split on left and right until min size
reached
```

BSPTree

`private int dungeonWidth;`
`private int dungeonHeight;`
`private BSPLeaf root;`
`private int[][] tileMap;`
	`//Dungeon width`
	`//Dungeon height`

`public int getDungeonWidth()`	`// get width`

public int getDungeonHeight()	// get height
public BSPLeaf getRoot()	// get root leaf
public int[][] getTileMap()	// get tile map
public void loadMap()	// Generate random dungeon
private void placeHallways(BSPLeaf leaf)	// Place hallways
private void placeRooms()	// Place rooms
public void adjustMap(int hallwayWidth)	// Smooth out map
private boolean makeSurroundCondition(int r, int c, int tileType)	// Helper method to make boolean condition to check surrounding tiles
public void placeWalls()	// Place walls
public void placeSpecial()	// Place special tiles

```
public int getDungeonWidth()
     return dungeonWidth

public int getDungeonHeight()
     return dungeonHeight

public BSPLeaf getRoot()
     return root

public int[][] getTileMap()
     return tileMap

public void loadMap()
     placeHallways()
     placeRooms()
     adjustMap(3)
     placeWalls()
     placeSpecial()

private void placeHallways(BSPLeaf leaf)
     connect the center of leaf's left and right subleaves
     recursively call placeHallways on left and right
```

```
private void placeRooms()
     for every leaf in root’s tree
          create a new room
          seet the leaf’s room to the newly created room

public void adjustMap(int hallwayWidth)
     int repeat = 25

     while (repeat > 0)
          for every room
               check surroundings
               if any side has a malformed wall
                    extend that side
          repeat--

private boolean makeSurroundCondition(int r, int c, int tileType)
     return tileMap[r + 1][c] == tileType || tileMap[r - 1][c] ==
tileType || tileMap[r][c + 1] == tileType || tileMap[r][c - 1] ==
tileType || tileMap[r + 1][c + 1] == tileType || tileMap[r + 1][c -
1] == tileType || tileMap[r - 1][c + 1] == tileType || tileMap[r -
1][c - 1] == tileType;

public void placeWalls()
     for every tile, check if there’s an adjacent room or hallway
tile
          if so, tile becomes a wall

public void placeSpecial()
     place spawn tile in first room
     place end tile in last room
```